PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

| | | |
|---|---|---|
| (51) International Patent Classification 4 : G06F 15/31, 15/347 | A1 | (11) International Publication Number: WO 90/00287<br>(43) International Publication Date: 11 January 1990 (11.01.90) |

(21) International Application Number: PCT/US89/02864

(22) International Filing Date: 30 June 1989 (30.06.89)

(30) Priority data:
214,665 1 July 1988 (01.07.88) US

(71)(72) Applicant and Inventor: MORTON, Steven, G. [US/US]; 39 Old Good Hill Road, Oxford, CT 06483 (US).

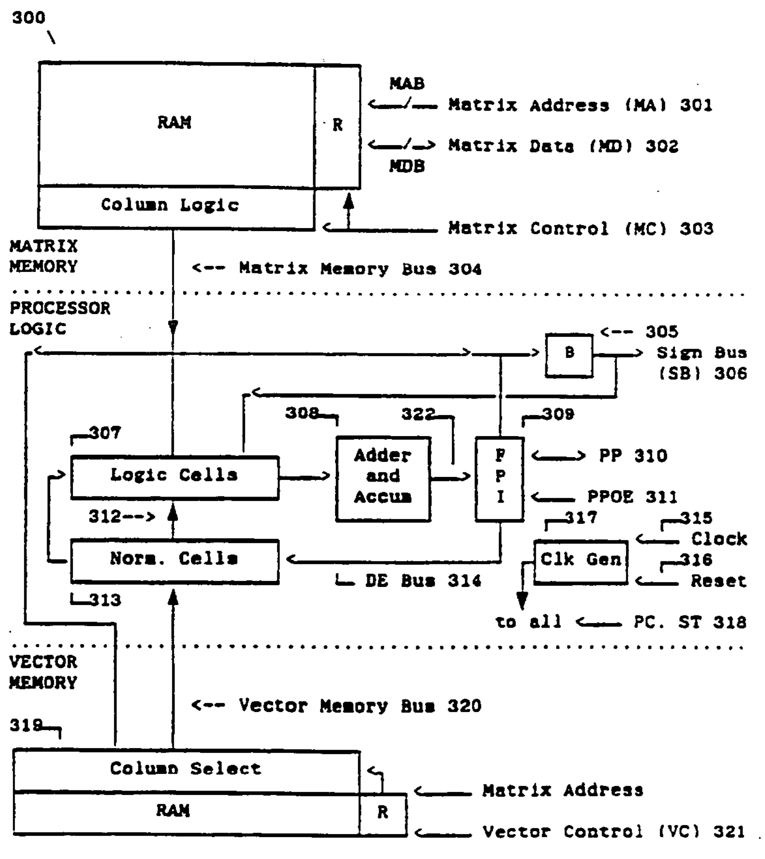
(81) Designated States: AT (European patent), BE (European patent), CH (European patent), DE (European patent), FR (European patent), GB (European patent), IT (European patent), JP, KR, LU (European patent), NL (European patent), SE (European patent).

Published
*With international search report.*

# (54) Title: INTELLIGENT FLOATING-POINT MEMORY

## (57) Abstract

The methods of operation and logic design of a digital, memory-plus-processor unit (203) are given. This unit both stores and multiplies matrices within it. Each unit stores one or more bits of each element of a matrix. Multiple units (202, 203) work together to give the precision of the matrix desired. Data may be represented in either fixed-point or floating-point format. Each unit combines a high capacity, very-wide-word memory (300) that stores one or more large matrices, a lower capacity memory (319) that is loaded from the high capacity memory and stores a vector or a smaller matrix, and highly parallel, simple processing logic (307) that feeds an on-chip, global adder and accumulator circuit (308).



## Description

## Intelligent Floating-Point Memory

### Technical Field

The invention relates primarily to semiconductor devices that multiply matrices using either fixed-point or floating-point arithmetic. Matrix multiplication is a fundamental operation that is required for digital signal processing, pattern recognition, graphics, and scientific-and-engineering computing. This operation may be advantageously implemented by the digital, parallel processing units described herein that combine a small amount of processor logic with a large amount of memory into a single unit in a modular fashion. Multiple such units work together to provide the performance and precision desired, and to handle more complex problems than a single unit can handle alone.

### Background Art

U.S. patent #4,777,614 (Ward) 11 Oct 1988, "Digital Data Processor for Matrix-Vector Multiplication", describes a two-dimensional, orthogonal, systolic array of bit-level processing cells that multiply matrices. The matrix must be fed into the array from one direction while the vector is fed in from an orthogonal direction.

U.S. patent #4,493,048 (Kung et al) 08 Jan 1985, "Systolic Array Apparatus for Matrix Computations", describes a two-dimensional, systolic array of inner-product-step processors. Data must be fed into the array for computations to be done.

U.S. patent #4,150,434 (Shibayama et al) 17 Apr 1979, "Matrix Arithmetic Apparatus", describes a pipelined system for performing matrix operations. Each arithmetic element handles an entire matrix element, rather than being distributed across multiple chips, and is not colocated with bulk memory for economy of communication.

U.S. patent #3,440,611 (Falkoff et al) 22 Apr 1969, "Parallel Operations in a Vector Arithmetic Computing System", describes a vector arithmetic. multiprocessor computing system for performing multiword-parallel vector operations such as sum reduction and search for largest. Each arithmetic element handles an entire matrix element. rather than being distributed across multiple chips. and is not colocated with bulk memory for economy of communication.

M. Duranton and J. A. Sirat. in "Learning on VLSI: A General Purpose Digital Neurochip". in the Proceedings of the International Joint Conference on Neural Networks. June 1989. page II-613 (abstract only). describe a chip that simultaneously multiplies multiple elements of a matrix and vector. and sums these products. The multiplication is done in several steps to minimize the amount of hardware required to build the multiplier. The matrix and vector are stored in two separate memories that are on the same chip as the arithmetic logic. The chip supports only 8-bit and 16-bit precision of the matrix. is specifically designed for connection to a single microprocessor (the Transputer). does not readily support its use with other identical chips to spread the bits of the matrix over multiple chips. and only performs fixed-point arithmetic.

The applicant of this application filed an international patent application. "An Intelligent Memory Chip for Matrix Multiplication". serial number PCT/US 88/04433. that describes how to perform matrix multiplication and convolution with fixed-point representation within multiple identical. specially configured memory chips.

## Disclosure of Invention

The object of the invention is to multiply matrices having either a fixed-point or a floating-point representation. The design is modular so that multiple units can work together to provide whatever precision is desired. Since complex problems are represented by large amounts of data. the bulk of each unit is memory which stores the data. Simple processing logic is placed on the same unit as the memory to minimize its complexity and cost. and to avoid transferring the data from the memory to distant processing logic. The interface to the memory is made as simply as possible. like the interface to common memory chips. which have no processing logic.

## Brief Description of the Drawings

The details of carrying out the invention are described by schematics. block diagrams. equations and methods in the following Figures. The Figures are according to this invention unless otherwise noted.

Figure 1 is equations for matrix multiplication (prior art).

Figure 2 is a block diagram of the intelligent floating-point memory module using multiple intelligent floating-point memory units.

Figure 3 is a block diagram of the intelligent floating-point memory unit.

Figure 4 is a diagram showing the spatial and temporal implementation of the multiplier.

Figure 5 is a timing diagram for the clock generator.

Figure 6 is a table of data storage formats for a row of the matrix memory.

Figure 7 is a block diagram of the matrix memory.

Figure 8 is a block diagram of the vector memory.

Figure 9 is a block diagram of the normalizer cell.

Figure 10 is a table of functions for the normalizer cell.

Figure 11 is a block diagram of the processor logic cells and adder-and-accumulator.

Figure 12 is a block diagram of one processor logic cell.

Figure 13 is a block diagram of the 8-port floating-point vector accumulator and interface unit.

Figure 14 is a block diagram of the 8-port mantissa section.

Figure 15 is a block diagram of the 8 by 40-bit weighted adder.

Figure 16 is a block diagram of the bus interface.

NOTE: For the convenience of the reader. all reference numbers are tied to the Figure numbers. Reference numbers are of the form XXYY. where XX is the Figure number. exclusive of any letter suffix. and YY is a reference number within that Figure.

## Modes for Carrying Out the Invention

Figure 1 shows the equations that define the multiplication of matrices with four columns. In equation 1.1. a large input matrix [A] has four columns as required by 3-D graphics. and has as many rows as are required to store the many vectors that represent 3-D objects. One hundred thousand to a million vectors may easily be required. The precision of each element of each of these vectors is typically 32 bits. in either fixed-point or floating-point format. This invention supports both formats.

A transformation matrix [B] is multiplied times each of many of the row-vectors in the input matrix. The input matrix may represent multiple objects. each having its own transformation matrix. in which case the transformation matrix must be changed depending upon which object is being handled. It is assumed. however. that the vectors of each object are grouped in adjacent rows of the input matrix in order to minimize the number of times that a new transformation matrix must be loaded.

The output matrix [C] is the conventional product of the input matrix and the transformation matrix. The invention provides for the storing of the output matrix within the intelligent memory chips without disturbing the external data bus or processor. in which case successive transformations may easily be computed.

It is important to note that this invention may be applied to matrices with any number of columns. The preferred embodiment is given with four columns because 3-D graphics is a large market which requires four columns. but there is no limitation to four columns. It is also important to note that this invention applies to the multiplication of matrices regardless of their application. The choice of graphics. signal processing. scientific computing or whatever has no bearing upon the principle of operation.

The method used by the invention to multiply the matrices is to decompose the operation into a sequence of simpler operations. Each row of the input matrix is handled in turn. As shown in equation 1.2. the first row of the input matrix is multiplied by the transformation matrix to produce the first row-vector of the output matrix.

The multiplication of a row-vector times a transformation matrix is further decomposed as shown in equation 1.3. In this invention. a row-vector of the input matrix is multiplied by the first column of the transformation matrix to produce the first element. a0. of the row-vector of the output matrix. The same row-vector of the input matrix is then multiplied by the second column of the transformation matrix to produce the second element of the row-vector of the output matrix. This process is repeated for the remaining columns of the transformation matrix. The next row-vector of the input matrix is then handled. and so on.

Figure 2 shows the block diagram of an intelligent floating-point memory module. If 32-bit precision of the input matrix is desired and eight intelligent floating-point memory chips as 202 and 203 share the load. then each unit stores four bits of each element of the matrix. Each unit is typically implemented as a single chip. so the words 'chip' and 'unit' will be used interchangeably from here on. These chips are controlled by a control unit 208 that is commanded by a host processor via the microprocessor (uP) data bus 210 to multiply a matrix composed of a group of row-vectors times a transformation matrix composed of four column-vectors.

Since each intelligent floating-point memory unit computes only a part of a product. the partial products as 206 and 207 must be combined to form a complete product. The floating-point vector accumulator and interface unit 209 combines these partial products. Note that scale factors are associated with each of the partial products as 206. Like the use of ordinary 4-bit memory chips. the memory data bus 201 places four bits of each element of the matrix in each four-bit chip. in which case bits 3:0 are placed in chip 203 and bits 31:28 (i.e., 31. 30. 29 and 28) are placed in chip 202. Since the least significant bit of chip 203 has a weight of 2^0. its partial product (PP) has a scale factor of 2^0. Likewise. chip 202 has bit 28 as its least significant bit. in which case its partial product 206 has a scale factor of 2^28.

The invention is not restricted to the use of four bits in each chip. Four is given for the example because many applications require 32 bits of precision and it is often convenient from a packaging point of view to use eight chips with four bits each.

The spatial significance of the chips is indicated by the slice type (ST) input. Only the most significant slice. chip 202. has the slice type equal to 1 since it stores the sign bit (assuming two's complement arithmetic): the other chips as 203 have slice type = 0.

As will be explained. the partial product line as 206 carries a variety of information between the vector accumulator unit 209 and the intelligent memory chips. The sign bus 204 provides all intelligent memory chips with the sign bits of the matrix elements being used for a particular operation. These sign bits are located in the most significant slice. This bus has four bits for chips that operate upon matrices with four columns.

Figure 3 shows the block diagram of the intelligent floating-point memory chip. It has three main sections: a matrix memory 300, processor logic and a vector memory 319. A matrix is loaded into the matrix memory via the matrix address 301, matrix data 302 and matrix control 303 lines. A transformation matrix is loaded into the vector memory from the matrix memory under control of the vector control lines 321. Information from both memories enters the processor logic and is processed. In the preferred embodiment, MDB, the number of matrix data bits, is 4, and MAB, the number of matrix address bits is 9, corresponding to a memory with multiplexed row and column addresses and a total of 1M-bits of storage (256K * 4).

The processor logic includes a buffer 305 for the sign bus 306, an adder and accumulator 308, a floating-point interface (FPI, 309) and a clock generator 317. The floating-point interface produces the partial product lines (PP, 310) which it drives if the partial product output enable line (PPOE, 311) is asserted and an output is required. The interface also receives information from the partial product lines and conveys it to the delta exponent bus (DE, 314). When the chip is designed to handle matrices with four columns, there are four logic cells within block 307 and four normalizer cells within block 313. The clock generator 317 receives the clock 315 and reset lines 316. The processor control (PC) and slice type (ST) lines drive all of the processor logic.

Figure 4 shows the spatial and temporal implementation of a multiplier as used by this invention. Assuming 32-bit by 32-bit operation for sake of example, the product can be implemented with thirty-two small multipliers, each handling 4 columns of bits and 8 rows of bits of the computation. Each binary product and sum can be handled by a tiny, 1-bit cell as P 400. Note that some of the small, multi-bit multipliers as 402 operate only upon unsigned bits (A3:0 and B7:0), whereas others as 401 have a combination of signed and unsigned bits (A31:28 and B7:0) depending upon whether bit A31 or bit B31 is being handled. Both A and B inputs to multiplier 403 are signed. A flexible means of specifying which combination of bits is to be handled by each chip is provided by the invention so that a single type of chip can handle all cases.

The invention handles each 4-bit wide slice of the multiplier in a single chip, in which case 8 chips provide 32 bits -- the precision required for graphics. Each 8-bit-high slice is provided in time by passing successive sets of data through a single 4-bit by 8-bit multiplier. More or fewer cycles may be used to provide more or less precision. Data is placed in the chips so that the spatial dimension determines the precision of the row-vector of the input matrix, while the temporal dimension determines the precision of the column-vector of the transformation matrix.

Note that the amount of multiplier logic for each product on each chip is only 1/32 of a 32-bit by 32-bit multiplier. As a result, the manufacturing yield is high since the amount of logic is very small. Even less logic may be used by using more cycles to compute the product. An 8-bit-high slice was chosen to provide a particular level of performance with a clock rate that is easily achieved with fabrication capability in 1988.

Similarly, 32-bit precision of the row-vector of the input matrix could be provided by 4 chips with 8-bit wide slices rather than having 8 chips with 4-bit wide slices. The choice is purely an engineering one based upon the configuration that best suits an application. Note, however, that as the width increases, the amount of logic increases, reducing yields, increasing the number of connections on each chip and increasing costs. The yield problem, while small, could be handled by placing spare multipliers on a chip and connecting them depending upon where defects fall.

The total amount of multiplier logic per chip is thus four, 4-bit by 8-bit multipliers, equivalent to one 8-bit by 16-bit multiplier. The adder and accumulator reduces the data rate on the partial product pins by combining the results of four 8-bit cycles. The clock generator is required to provide the multi-cycle operation for each product.

The type of memory technology used for the matrix memory 300 and the vector memory 319, and its storage capacity, are chosen to suit the application and the state of technology; they are not fundamental to the invention. Typically, the capacity of the matrix memory will be many times greater than the capacity of the vector memory.

Figure 5 shows the signals generated by the clock generator 317 in response to clock 315 and reset 316. Reset 316 is asserted upon system initialization to synchronize the operation of all intelligent memory chips. Each chip then acts as a simple finite state machine. The timing signals shown may be generated by many shift register or counter means as are known in the art. Four cycles are shown as T0 (507), T1 (508), T2 (509) and T3 (510), consistent with the use of four, 8-bit temporal slices to accomplish a 32-bit multiplication. Other precisions can be provided by varying the number of cycles, either dynamically (under program control) or statically (at hardware design time).

FClk 502 is the "fast clock". MClk 504 is the "medium clock" (MClk phase 1), and SClk 506 is the "slow clock" (SClk phase 3). The fast clock is the rate that successive 8-bit portions of one column-vector of the transformation matrix are handled. The medium clock is the rate at which partial products are conveyed between chips. The slow clock is the rate at which complete, 32-bit products are computed. The bits shown for Tn (n = 0 to 3) are at the output of the Vector Memory. The bits shown for T0.1 (511) and T2.3 (512) are at the output of the Partial Product bus.

Figure 6 is a table of data storage formats for the matrix memory 300. The processor logic operates upon data that is read at one time from a row of the matrix memory, so there are constraints upon how data is stored. These constraints are key to the operation of the invention.

Note that the many row-vectors of the input matrix (shown under matrix format) are stored efficiently, where each bit of each element is stored only once in a group of intelligent memory chips. However, the few column-vectors of the transformation matrix are passed through the matrix memory to the vector memory and are stored redundantly. (Close attention must be paid to which vector is which to avoid confusion.) Since each intelligent memory chip uses the entire column-vector, and there are eight intelligent memory chips in the module shown in Figure 2, there is eight-fold redundancy in the storage of the column-vector. This redundancy of storage eliminates a massive wiring problem that would arise if the column-vector were stored only once and wired to all of the chips that need it.

Assuming the placement of four bits in the matrix memory 300 (MDB = 4 in Figure 3) of each element of the input matrix, then the matrix memory would be implemented from four planes (700, 701, 702 and 703) of memory cells. Note also that while a single bit is written into each plane from the matrix data bus 302, as is conventional for a by-4 memory. Further assuming that matrices with four columns are operated upon by the intelligent memory chip, then four columns from each plane are selected simultaneously by the multiplexer 706 and connected to the matrix memory bus 304. Each bit is the same weight of bit for each of the four columns of the row-vector as shown in Figure 6. (This is an important point.)

For example, the data that may be present on the matrix memory bus in chip 203 in Figure 2 is:

```
Matrix Memory Bus B: 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

Matrix format:
            Row K. B:  3  2  1  0  3  2  1  0  3  2  1  0  3  2  1  0
            Column:    3  3  3  3  2  2  2  2  1  1  1  1  0  0  0  0
```

Vector format:

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Col L, B: | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |
| Row: | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Similarly, the data that may be present on the matrix memory bus in the next most significant chip is:

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Matrix Memory Bus B: | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |

Matrix format:

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (higher) | Row K, B: | 7 | 6 | 5 | 4 | 7 | 6 | 5 | 4 | 7 | 6 | 5 | 4 | 7 | 6 | 5 | 4 |
| | Column: | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Vector format:

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (same) | Col L, B: | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |
| | Row: | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Figure 7 shows that each memory plane as 703 has sense amplifiers and column read/write logic 704, as is common for a high speed RAM. The outputs of the sense amplifiers are also connected to an output register and multiplexers 706 to store the entire row of data and provide the multi-bit access described in Figure 6. The column write logic is also connected to an input register 705 that operates in a read-modify-write mode where multiple bits may be changed each time that a row of the memory is written. (If each of the four planes of the RAM stores 256K bits and is a square array of storage cells, then there are 512 sense amplifiers per plane and n = 512/4 = 128.) This input register allows multiple results to be accumulated and written into one row of the memory at the same time on an infrequent basis, rather than requiring memory write cycles to be performed at the same rate as vector dot-products are computed. The choice of the bits to be loaded is made by the matrix address lines, and the choice of loading the register rather than the memory is made by the matrix control lines.

Figure 8 is a block diagram of the vector memory 319. A typical minimum configuration would be 16 words by 32 bits so that it can store a 4-by-4 transformation matrix with 32-bit precision. Note, however, that each word has 8 bits from each of the elements in the four rows of a column-vector, rather than the 32 bits of a single element. This format is required because the four logic cells 307 work on 8 bits of each element of a column-vector at a time. A sequence of four words provides the 32 bits of each of the four elements.

Note that:

1. A column-vector from the vector memory is multiplied by a row-vector from the matrix memory.

2. The connections from the Matrix Memory Bus to the Input Bus are:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Input Bus B: | 31:28 | 27:24 | 23:20 | 19:16 | 15:12 | 11:08 | 07:04 | 03:00 |
| Matrix Memory Bus B: | 15:12 | 15:12 | 11:08 | 11:08 | 07:04 | 07:04 | 03:00 | 03:00 |
| Write Column Select: | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

3. The content of the Vector Memory Bus versus time is: T0 - B7:0. T1 - B15:8. T2 - B23:16. T3 - B31:24 from each of the four rows of the column-vector being processed.

Unlike the matrix memory 300 that is accessed infrequently because of the storage capacity of the output register 706, the vector memory is accessed often, at the fast clock rate. This speed difference is reasonable since the matrix memory is large and presumably slow, whereas the vector memory is small and presumably fast. However, the speed requirements of the vector memory can be reduced by increasing its width in proportion to a reduction in its height (the number of words). In this case the sense amplifiers 805 would read additional columns and a set of multiplexers (not shown) would select some of them for passage to the vector memory bus.

The write logic 806 provides the loading of sections of the vector memory from the matrix memory. The read logic 807 includes registers that allow the definition of a set of locations in the vector memory to be accessed cyclically for repetitive use of the transformation matrix.

Writing into the vector memory from the matrix memory takes two cycles per word since a word is 32 bits wide but the matrix memory bus 304, which supplies the data, is only 16 bits wide. It thus takes 32 cycles to load the vector memory with a 4-by-4 transformation matrix, so it is desirable to minimize the frequency with which it is loaded to minimize the amount of time that no computations are being performed. This problem can be alleviated by increasing the amount of storage and providing dual-port storage cells so that one transformation matrix may be loaded while another is being used. Alternatively, the matrix memory bus could be made wider.

The method for loading the vector memory from the matrix memory is:

1. Select a row of the matrix memory with Matrix Address and load the

row into the Output Register. This operation, like any operation using Matrix Address or the matrix memory, preempts use of the matrix memory.

2. Select a starting point in the Output Register with Matrix Address. This point must be bit 0 of the column-vector.

3. Select a starting address in the vector memory with Matrix Address. This point must be bit 0 of the column-vector.

4. Initialize a counter, N, to 0.

5. Move bits [(3:0) + 8N] of each row of the vector from the Output Register to the even nibbles (B: 3:0, 11:8, 19:16, 27:24) of the vector memory.

6. Advance to the next point (bit(4 + 8N)) in the Output Register.

7. Move bits [(7:4) + 8N] of each row of the vector from the Output Register to the odd nibbles (B: 7:4, 15:12, 23:20, 31:28) of the vector memory.

8. Advance to the next point (bit(8 + 8N)) in the Output Register.

9. Advance to the next row (bit(8 + 8N)) of the vector memory.

10. N = N + 1. If N < 4 then go to step 5 to handle successive groups of 8 bits of the Vector.

11. Done. The new vector is ready for use and the next may be loaded.

Figure 9 is a block diagram of the normalizer cell. The table of functions of the normalizer cell is given in Figure 10. The basic idea is that each vector mantissa may be shifted to the extent necessary to line up the binary point of all of the products of the mantissas. No shifting is performed for the product with the largest exponent; all others are shifted to match it, where the shift reduces the magnitude of the product. Low order bits of a vector element may thus be lost, but this produces an error in the sum of products that is very nearly the same as produced by losing bits when adding a series of products with conventional devices. However, if these small errors are significant, then the product can be computed with additional precision. This is possible because all bits of each vector element are available in each intelligent memory chip. Any increase in precision (beyond eight bits) is in direct proportion to the reduction in performance from handling the additional bits.

The shifting of the mantissa is implemented in two stages. Shifts of 8 bits are handled by loading different registers as 905, 906, 907. Shifts of 1 to 7 bits are handled by multiplexer 901 during the calculation of the product. Unlike conventional barrel shifters that require large number of gates to implement large shifts, the structure in Figure 9 requires relatively little logic because the byte-serial architecture allows the selective loading of register 2 (905), register 1 (906) or register 0 (907) under control of the SC (shift control) block 900.

Note if Figure 9 that the bit assignments for the four cells are:

Cell 0: FPI Bus B4:0, Vector Memory Bus B7:0, Shifter Bus B7:0.
Cell 1: FPI Bus B9:5, Vector Memory Bus B15:8, Shifter Bus B15:8.
Cell 2: FPI Bus B14:10, Vector Memory Bus B23:16, Shifter Bus B23:16.
Cell 3: FPI Bus B19:15, Vector Memory Bus B31:24, Shifter Bus B31:24.

The eight OR gates 908 are used to set the hidden bit of the mantissa as the data is being loaded into registers 2 to 0. The hidden bit is set only if the vector element is not zero as indicated by the DE (Delta Exponent) bus 314. In the IEEE single precision floating-point format, bit 22 is the hidden bit, corresponding to bit 6 of the OR bus 909.

Figure 11 is a block diagram of the floating-point processor logic. There are four processor cells, as 1102, that perform multiplication. Each receives four bits from the matrix memory bus and eight bits from the shifter bus 312. The products from these multipliers are summed and stored by the adder/register 1103. The sum of products is added by the ALU 1104 to the accumulated sum of products stored by the register 1105. The accumulated sum of products is shifted by 8 bits per cycle, corresponding to the use of successively higher weight sets of eight bits from the shifter bus. The final sum of products is stored in the register 1106 and passed in two sets of bits to the floating-point interface 309 by the multiplexer 1107 via the Internal Partial Product data lines 322.

Note in Figure 11 that:

1. The assignment of the Matrix Memory Bus to each Cell's input "A" is:
   Cell 0 B0 to B3 is Matrix Memory Bus B0 to B3.
   Cell 1 B0 to B3 is Matrix Memory Bus B4 to B7.
   Cell 2 B0 to B3 is Matrix Memory Bus B8 to B11.
   Cell 3 B0 to B3 is Matrix Memory Bus B12 to B15.
2. The ALU function is Σ = A during the first cycle for each column-vector, otherwise Σ = A plus B.
3. The mantissa sign (MS) input to each Cell from the sign bus 306 is used only for floating-point. It is the product of signs of the row-vector and column-vector elements it is working on.

Also note for single precision floating-point that the mantissa has only 23 bits, rather than 32, hence three cycles (each handling eight bits), rather than four cycles, are required to compute a product. However, four cycles are needed per product to pass data between the intelligent memory chips and the vector accumulator chip 208, so no increase in performance is realized. Thus an additional eight bits of precision may be provided without degrading performance, but this is not shown.

Figure 12 is the block diagram of the floating-point processor cell. It includes a hidden-bit register 1203 that specifies the bit position of the mantissa hidden bit for each element of the input matrix. This bit is set only for bit 22, a bit that occurs in a single intelligent memory chip, whereas the hidden bit for the column-vector mantissa occurs in each chip since the entire column-vector is stored in each chip. (When a non-zero number is in floating-point format, one bit of the mantissa does not need to be stored because it is always a 1; it is called the "hidden bit".)

The mask register 1202 is used to inhibit (turn off) the bits that are not in the mantissa of an element of the input matrix. For the IEEE single precision format, bits 21:0 are in the mantissa and the corresponding mask bits would be 1's; the remaining mask bits would be 0's.

These registers must be loaded via the matrix memory under the control of the processor control lines 318 before computations begin.

Note that only the logic for single precision floating-point is described. The extension to double precision requires a doubling of the width of the vector memory and the normalizer, and twice as many intelligent memory chips to provide 64-bit matrix precision. The mask register and hidden register are sufficiently general so that no changes are required; they would simply be loaded with appropriate data.

The "F" gates as 1204 handle masking and hidden bit injection. Each gate has three, 1-bit inputs and a 1-bit output. The function is: {[Data Register AND Mask Register] OR [Hidden Bit Register AND (NOT Matrix Zero)]}.

For floating-point operations, the multiplier 1206 operates only on positive values since floating-point mantissas are stored in sign/magnitude format. For fixed-point operations, a signed multiplier is used whenever a sign bit is present, as indicated by the slice type and clocks.

For floating-point operations, the sign block, "S", 1208 receives the sign of the matrix mantissa from the sign bus, and the sign of the column-vector mantissa from the vector memory bus. If the sign of the product is negative, then the ALU 1209 complements the output from the multiplier 1206. Otherwise, and for fixed-point operations, the data passes unchanged.

For fixed-point arithmetic, the normalizers act as though they were invisible, the mask registers are set so that all bits are used, and the hidden bit registers are cleared. However, the number of cycles to produce a floating-point product is much greater than the number of cycles to compute a fixed point product because information must flow from the intelligent memory chips to the vector accumulator chip and back. However, the rate of dot-product computation is the same for both fixed-point and floating-point.

For fixed point arithmetic, the multiplier 1206 is controlled by two signals so as to implement the correct section of a 32-bit by 32-bit multiplier. The spatial selection is made by slice type (ST) which distinguishes the most significant slice of the matrix from lower significance slices. The temporal selection is made by T0 which distinguishes the most significant slice of the vector from lower significance slices. The choice of timing signal T0 is a result of the pipelining of the structure; it is the signal that is asserted when the most significant bits of the vector are flowing through the multiplier.

The data register 1201 and mask register 1202 are loaded when commanded by the processor control inputs to the chip, whereas the vector register 1205 is loaded whenever run (one of the processor control inputs) is asserted, which is whenever multiplications are being performed.

Figure 13 is a block diagram of the floating-point vector accumulator and interface chip 209. The exponent section 1303 performs the exponent arithmetic required by the method for performing floating-point arithetic described herein. The mantissa section 1306 checks for matrix mantissas that are zero and sends this information to the exponent section via the exponent bus 1305. The presence or absence of a zero mantissa for each element is sent to all intelligent memory chips via the exponent section. The mantissa section receives exponent information from the exponent section and includes normalization logic as is known in the art to provide the correct floating-point format of the final vector dot-product.

The mantissa section 1306 also combines the partial products into a complete product. The bus interface 1307 routes data between the intelligent memory chips via the MD bus 201, the mantissa section and the microprocessor data bus 210. A clock generator 1308 provides timing signals; it is synchronized by reset 1310 to the clock generator as 317 in each intelligent memory chip to facilitate the passing of data between chips. The reset signal can be shifted in time between the vector accumulator chip and the group of intelligent memory chips should changes in the pipelines require it.

Figure 14 is a block diagram of the logic within the mantissa section that combines partial products. The speed requirements of the weighted adder 1406 have been minimized by buffering the partial products as 1400 in such a way as to present the full 40 bits of each partial product to the adder at the same time. The first bank of registers as 1403 delays the partial product so that the second bank of registers as 1404 can capture the entire partial product and hold it for an entire slow clock period.

The output 1407 of the adder 1406 passes through a register 1408 to a rounding unit 1409. For graphics, the 68-bit product 1407 would typically be truncated or rounded to 32 bits. The choice of manipulation provided by the round unit depends upon the needs of the application. A full precision product could be passed if it were desired to combine the outputs from multiple vector accumulators in order to handle matrices with more than 4 columns. The output of the round unit is captured by an output register 1410.

Figure 15 is the block diagram of the 8-port by 40-bit weighted adder 1406. It is composed of four adder/registers 1502 to 1505 in the first layer. (An adder/register is an adder followed by a pipeline register.) Each combines a pair of partial products whose significance differs by a factor of 16, consistent with each intelligent memory chip handling 4 bits. The least significant partial product, PP0 1500, feeds the 40 AL ("A" least significant) inputs to adder/register 1502. The most significant bit is used to extend the product by 4 bits at the AM ("A" most significant) inputs. The four least significant bits of the other input to the adder/register are set to zero at the BL input. PP1 1501 is fed into the 40 most significant bits at the BM input. The adder/register then computes a 44-bit sum. The other adder/registers in the same level operate similarly.

The second layer of adder/registers 1507 and 1506 combine the outputs from the first layer of adder/registers. Adder/register 1508 in the third layer combines the results from the second layer and produces sum out 1509.

Figure 16 is a block diagram of the bus interface 1307. It has two driver/receivers 1600, 1602 to communicate with the external busses, MD and I/O. The novel feature of the interface is its eight sets of multiplexers, where each set has four multiplexers as 1606 to 1609. These multiplexers provide the loading of each 4-bit nibble of the vector into each of 8 intelligent memory chips.

The method for loading a vector into the matrix memory is described below. The objective is to load all bits of each element of the column-vector into each intelligent memory chip, whereas only four bits of each element of the matrix are loaded into each chip.

The following procedure is executed for each 32-bit element (or row) of each column-vector. It is controlled by the control unit 208 in the intelligent floating-point memory module shown in Figure 2. The control unit is instructed by a microprocessor to perform the operation and to store the vector at a particular address. The procedure is:

1. The vector element is loaded into the Write Vector Register in the vector accumulator chip. The element may come from the microprocessor data bus, the memory data bus or the dot product output.

2. Under control of matrix control and matrix address, the input registers in each intelligent memory chip are loaded from the row of the matrix memory that will store the vector element.

3. No writing to this row of the matrix memory may be performed while the following steps take place.

4. For each 4-bit nibble, where nibble N comprises bits 4N+3 to 4N, and N = 0 to 7:

4a. The selected nibble from the vector write register in the vector accumulator chip is replicated eight times by the multiplexers in the bus interface logic in the vector accumulator chip to fill the entire 32-bit word. The selection of the nibble is made by the vector accumulator control (VAC) lines.

4b. The word containing the replicated nibble is passed to the memory data bus and received by the intelligent memory chips.

4c. This nibble is written into the input register in the intelligent memory chips, where the position in the register is selected by the memory address lines. The position must be selected in accordance with the "table of data storage formats in a row of the matrix memory" (Figure 6).

5. The input register is written back into the row of the matrix memory. Only the bits that were updated by the nibbles of the column-vector are changed.

6. Full use of the matrix memory may resume.

7. The next element of the vector may now be handled as described above.

In a typical operation, a matrix of row-vectors is multiplied by a transformation matrix. Each product is available at the output of the mantissa section 1306 and flows to the register 1601 in the bus interface

1706. The output of register 1601 is turned on, all other devices connected to the bus 1605 are turned off, and the product flows through driver/receiver 1600 to the MD bus for loading back into the intelligent memory chips.

The intelligent memory chips may be read or written like common memory chips by the host processor via the microprocessor data bus that is connected to the I/O port of driver/receiver 1602. Data is written into the memory chips by turning on the receiver in driver/receiver 1602 and the driver in driver/receiver 1600. The situation is reversed to read data from the intelligent memory chips back into the microprocessor. The control unit 208 would receive a bank-select signal that it passes to the memory chips and the vector accumulator chip to determine when they should respond.

The use of multiple intelligent memory chips to perform floating-point operations is now given. First, the terminology for describing the operation of floating-point data is given below. As is common practice, each matrix element is the product of a mantissa and a power of 2. As is also common practice, the mantissa is held in sign/magnitude representation and the exponent is held in offset binary.

Let: [M] = [M0, ..., M(N-1)] and [V] = [V0, ..., V(N-1)].
where:

```
    Mk = MMantissa(k) * 2^MExponent(k) = MMk * 2^MEk
    Vk = VMantissa(k) * 2^VExponent(k) = VMk * 2^VEk
    1.0 <= magnitude of mantissa < 2.0, or magnitude = 0.
```

The conventional method for computing the floating-point product of a row-vector and a column-vector is given below. First, a running sum in floating-point format (generally an extended format) is initialized. Then, and this is a key point, the product of the mantissas of a pair of elements is computed in fixed point representation, and then a shift of the smaller of the product and the running sum is performed so that the exponent of the product and the running sum are the same, at which point the product is added to the running sum. When a series of products is complete, the result is renormalized. Note that this method works on one product at a time.

For C = [M] * $[V]^T$ (ignoring special cases):

```
1:  MT0 = 0; initialize MantissaTemporary0; accumulator
2:  ET0 = 0; initialize ExponentTemporary0; accumulator
3:  k = 0; initialize loop counter
```

```
4:
5:        MT1 = MMk * VMk: multiply mantissas in fixed-point
6:        ET1 = MEk + VEk: add exponents in fixed-point
7:        ET0 = larger of ET0 and ET1
8:        shift MT1 and adjust ET1 so ET1 = ET0: denormalize
9:        MT0 = MT0 + MT1: accumulate - add mantissas in fixed-point
10:       normalize MT0 and adjust ET0: renormalize
11:       k = k + 1
12:       If k < N Then GOTO 5
13:
14:  C = MT0 * 2^ET0
```

The computation method that is implemented by intelligent floating-point memory chips is given below. Since an intelligent memory chip adds several products simultaneously, the solution is to shift each element of the vector prior to computing the product. The products thus have a common exponent and may be added. Any number of products may be handled simultaneously by this method, although the preferred embodiemnt of the chip described herein operates upon only four.

This method is possible because the entire vector is stored in each chip, so all bits are available to perform the shift. The matrix, on the other hand, is spread over multiple chips and cannot be shifted without a serious interconnection problem between chips.

The details of this method are:

Let degree of parallelism = P = number of products computed in a group:

```
1:   MT0 = 0; initialize MantissaTemporary0; accumulator
2:   ET0 = 0; initialize ExponentTemporary0; accumulator
3:   k = 0; initialize loop counter
4:
5:        in parallel: E0 = MEk + VEk, ..., E(P-1) = ME(k+P-1) + VE(k+P-1);
sum the pairs of exponents in the group
6:        EP = Max(E0, ..., E(P-1)); find the largest exponent
7:        in parallel: S0 = E0 - EP, ..., S(P-1) = E(P-1) - EP
8:        in parallel: MT1 = [MMk * (VMk * 2^S0)] + ... + [MM(k+P-1) *
(VM(k+P-1) * 2^S(P-1))]; multiply and add at the same time
9:        normalize MT1 and adjust EP; renormalize
10:       add (MT1,ET1) to (MT0,ET0); accumulate as normal
```

SUBSTITUTE SHEET

```
11:       k = k + P
12:       If k < N Then GOTO 5
13:
14:  C = MTO * 2^ETO
```

The Partial Product (PP) Bus is used as follows:

Cycle 0: Matrix Data -> Vector Accumulator Chip from each Intelligent Memory Chip.

Cycle 1: PP19:0 -> Vector Accumulator Chip from each Intelligent Memory Chip.

Cycle 2: Delta Exponent -> all Intelligent Memory Chips in parallel from the Vector Accumulator Chip.

Cycle 3: PP39:20 -> Vector Accumulator Chip from each Intelligent Memory Chip.

The operations required to multiply a row of a matrix times a column of a vector are as follows. The operations would be highly pipelined and overlapped for efficient operation, but these complexities are ignored for simplicity of explanation. It is assumed that appropriate data has been loaded into the matrix memory from a host processor prior to beginning this procedure.

0. The Mask Register and the Hidden Bit Register are initialized in turn from the Output Register according to the desired floating-point format. For the IEEE 32-bit standard: (1) Mask Register bits 31-22 are cleared (sign bit and exponent bits) and bits 21-0 (mantissa or significand bits) are set, and (2) Hidden Bit Register bit 22 is set and all other bits are cleared. This initialization enables identical intelligent memory chips to provide the appropriate function for each bit.

1. The Output Register is loaded with a row of the Matrix Memory.

2. A portion of the Output Register is selected as the "current vector column".

3. The Vector Memory is loaded with the current vector column.

4. All bits of each element in the current vector column are sent to the Vector Accumulator Chip. They are stored in the Exponent Section.

5. The Output Register is loaded with a row of the matrix from the Matrix Memory.

6. A portion of the Output Register is selected as the "current matrix

row".

7. All bits of each element in the current matrix row are sent to the Vector Accumulator Chip. They are stored in the Mantissa Section.

8. The sign bit of each element in the current matrix row is sent to all Intelligent Memory Chips via the Sign Bus.

9. The Exponent Section in the Vector Accumulator Chip computes the exponents of the products, selects the maximum exponent, and computes the Delta Exponents - the difference between the maximum and each product's exponent.

10. The Mantissa Section in the Vector Accumulator Chip checks each matrix element for a zero value.

11. The Exponent Section in the vector Accumulator Chip checks each vector element for a zero value.

12. The Delta Exponents are sent to all Intelligent Memory Chips. The checks for zero for the matrix and vector are used to form the Delta Exponent.

13. The Vector Memory is loaded into Registers 2 to 0 in the Normalizer Cell under control of the Delta Exponent, shifting the vector mantissa by multiples of 8 bits. The Hidden Bit is not set if a vector element is zero.

14. Registers 2 to 0 in the Normalizer Cell are loaded into Registers 5 to 3.

15. The sign bit of each product is computed by the Processor Logic Cell.

16. The product of the matrix row and vector column are computed. The hidden bit of each matrix element is suppressed if the matrix element is zero. Each element of the vector column is shifted by up to seven bits by the Normalizer cell to complete the adjustment according to the Delta Exponent.

17. Go to the next vector column and/or matrix row.

## Industrial Applicability

The intelligent floating-point memory units described herein may be used for a wide range of applications requiring the rapid manipulation of large matrices of numerical data. These applications include digital signal processing, pattern recognition, three-dimensional graphics, and scientific-and-engineering computing.

Claims

1. An intelligent floating-point memory unit capable of performing a NumPoints (number of points - two or more) vector dot-product calculation upon data having a fixed-point or a floating-point representation and comprising:

(a) a matrix memory (300) capable of storing one or more matrices and one or more vectors, and receiving matrix control inputs and matrix address inputs, and driving/receiving NumMBits (number of matrix bits - one or more) matrix data pins, and comprising:

(a1) NumMBits memory-and-logic planes (703), each plane being connected to one bit of the matrix data bus and comprising:

(a1a) memory cells arranged in a multiplicity of rows and columns, and storing a row of the matrix in a row of memory cells,

(a1b) row-selection means for receiving the matrix address inputs and the matrix control inputs, and selecting one row of the memory cells,

(a1c) access means for reading information from, and writing information to, the memory cells, where information may be conveyed between a single column in a plane of the memory cells and one of the matrix data pins,

(a1d) an input register that stores a row of bits received from the access means for each plane of memory cells, that replaces those bits by one or more bits received from one of the matrix data pins, where the selection of each bit replaced is made by the matrix address inputs in conjunction with the matrix control inputs,

(a1e) an output register that stores a row of bits received from the access means and a multiplexer that selects NumPoints of these bits to be connected to the matrix memory bus,

(b) a vector memory (319) receiving the matrix memory bus, matrix address pins and vector control pins, producing the vector memory bus, and comprising:

(b1) multiple sets of words of storage, each word having NumPoints sections, each section having NumVBits (number of vector bits) bits, where NumVBits is an integer multiple of NumMBits, where data is loaded from the matrix memory via the output register and the matrix memory bus into the vector memory in such a way that for the bits that represent the mantissa:

(1) the bits within each section have ascending bit weight, (2) the same set of bit weights is used in each of the sections in a single word, and (3) as many words are used in each set as are required to store all of the mantissa bits of each element of the vector, in which case successive words have bits with ascending weights.

(c) processor logic receiving the matrix memory bus, vector memory bus, and processor control pins, where one of the processor control pins is "MSS slice" (aka "slice type"), and producing the partial product signals and the sign bus, and comprising:

(c1) a clock generator (317) that produces a repetitive sequence of timing signals as T0 to TL,

(c2) NumPoints normalizer cells (313), each receiving NumVbits of the vector memory bus and a portion of the DE bus, where each cell receives a different portion of each bus, and producing NumVBits of the Shifter bus, and comprising:

(c2a) a set of registers (as 905 to 907) that receive the vector memory bus, where the position that each portion of an element of a vector is loaded into the registers depends upon a value conveyed by the DE bus, and where registers that are not loaded are set to zero,

(c2b) a set of registers (as 902 to 904) that are loaded in parallel from the set of registers (as 905 - 907) and shift the element of the vector by NumVBits at a time where the least significant bits are lost first,

(c2c) a set of multiplexers (901) that receives the outputs of the 2 * NumVBits of the least significant bits of the registers (as 902 to 904) and produces NumVBits of the shifter bus, where choice of NumVBits of adjacent bits read from the registers depends upon a value conveyed by the DE bus.

(c3) NumPoints logic cells (307), each receiving one bit of the sign bus, NumMBits bits of the matrix memory bus and NumVBits of the shifter bus, where each cell receives a different portion of each bus and produces a product output, and comprising:

(c3a) a mask register (1202) receiving NumMBits of the matrix memory bus under command of the processor control pins,

(c3b) a hidden bit register (1203) receiving NumMBits of the matrix memory bus under command of the processor control pins,

(c3c) NumMBits gates (as 1204), each receiving one bit from the mask register, one bit from the hidden bit register and a signal, Matrix Zero, conveyed by a state of the DE bus, where the function of each gate is:

{[Data Register AND Mask Register] OR [Hidden Bit Register AND (Not Matrix Zero)]},

(c3d) a multiplier that forms the product of its gated matrix memory bus input and its shifter bus input. and produces the product output.

(c3e) an ALU (1209) that passes the output of the multiplier if the sign of the overall product is positive or complements the output if the sign is negative,

(c4) an adder and accumulator (308) receiving the product outputs from the logic cells and producing the internal partial product output. and comprising:

(c4a) an adder/register (1103) that sums the product outputs.

(c4b) an ALU (1104) and register (1105) that sum the adder (1103) output over time. where the weight of the adder output in each successive cycle is 2^NumVBits times the weight of the preceeding cycle. where the weight 1 when a new summation starts.

(c4c) a register and multiplexer (1106) that receives the completed sum from the ALU (1103) and register (1105). where the multiplexer passes portions of the output of the register to the internal partial product in sequential cycles,

and (d) a floating-point interface (309) that (1) receives the matrix memory bus and sends it on the partial product bus, (2) receives the internal partial product and sends it on the partial product bus. and (3) receives the partial product bus and sends it as the DE bus.

2. A method for computing the dot-product of vectors represented in floating-point format, comprising a series of steps:

Step 1: For vectors [M] and [V]. each having N elements. computation of the set of the sums of exponents, where S(i) = [exponent of M(i)] plus [exponent of V(i)], for i = 0 to N-1.

Step 2: Selection of the largest exponent, LE, from the set of sums of exponents.

Step 3: Calculation of the set of differential exponents where DE(i) = LE - S(i). for i = 0 to N-1.

Step 4: Reduction in the magnitude of the mantissa of V(i) by 2^DE(i). where the scaled mantissa of V(i) = mantissa of V(i) / 2^DE(i). for i = 0 to N-1.

Step 5: Calculation of the sum of the products of the mantissas of the M vector and the scaled mantissas of the V vector, where Sum = [mantissa of M(0) * scaled mantissa of V(0)] + .... + [mantissa of M(N-1) * scaled mantissa of V(N-1)].

Step 6: Normalization of the Sum, including calculation of its exponent.

Step 7: Addition of LE to the exponent of the normalized Sum.

Step 8: Exit.

3. A nibble-serial, element-parallel method for multiplying a row-vector times a column-vector, comprising:

(a) a data structure, comprising:

(a1) two or more vectors, each with NumPoints (number of points) elements,

(a2) one or more row-vectors [R(i)], each of whose elements has M * NumMBits (NumMBits = number of matrix bits) bits of precision in two's complement notation,

(a3) one or more column-vectors [C(i)], each of whose elements has V * NumVBits (NumVBits = number-of-vector-bits) bits of precision in two's complement notation,

(b) an intelligent memory module capable of synchronizing all of its members, comprising:

(b1) M identical storage-and-processing units, aka intelligent memory units, each unit containing a cycle counter and operating upon NumPoints (number of points) elements of a vector, storing a group of NumMBits (number of matrix bits) adjacent bits of each element of the row-vector, where unit 0 stores bit 0 and groups of ascending bits are stored in ascending units, storing all bits of each element of the column-vector, operating upon NumVBits (number of vector bits) adjacent bits of each element of the vector during a single cycle as selected by its cycle counter, producing a series of sub-partial products, and producing a partial product output following the last cycle.

(b2) one global combinatorial unit, aka a vector accumulator unit 208, that receives the partial products from each of the M intelligent memory units and combines these partial products according to their respective significance to form a complete product.

and (c) a sequence of steps, comprising:

Step 0: Initialization of a cycle counter,

Step 1: Initialization of a sub-partial product accumulator in each unit,

Step 2: For each of the M units, for each of the NumPoints, selection of NumVBits of the vector by the cycle counter, and computation of its NumMBits matrix bits and its NumVBits vector bits,

Step 3: For each of the M units, summation of the results of the multiplications and addition of the summation to the sub-partial product accumulator according to the significance of the summation of the results of the multiplications,

Step 4: Advance to the next cycle, and if there are more cycles then go to Step 2,

Step 5: For each of the M units, assignment of the sub-partial product to the partial product, and transmission of each partial product to the global combinatorial unit,

Step 6: Combination of the M partial products according to their significance and production of the final product,

Step 7: Exit.

4. A vector accumulator and interface unit (208) capable of operating upon words with M bits where M = K * NumMBits and K is an integer, comprising:

(a) an arithmetic section (1306) comprising:

(a1) a weighted adder tree having K multi-bit inputs and producing a sum output, where the inputs are scaled prior to being added so that the weight of the least significant bit of each of these K inputs increases by 2^NumMBits and the weight of the least significant bit of the least significant input is 1,

(a2) a round unit (1409) producing a rounded output and capable of passing a portion of sum outpu- to its rounded output,

(b) a bus interface (1307) receiving a select input and comprising:

(b1) a write register (1601) capable of receiving the rounded output and a value from an external data bus,

(b2) K sets of multiplexers with NumMBits multiplexers in each set (as 1606 to 1609) and capable of replicating any group of NumMBits adjacent bits from the output of the write register throughout an M-bit word, where the least significant bit in a group is bit number (J * NumMBits) for J = 0 to K-1 and the group is chosen by the select input.

Figure 1

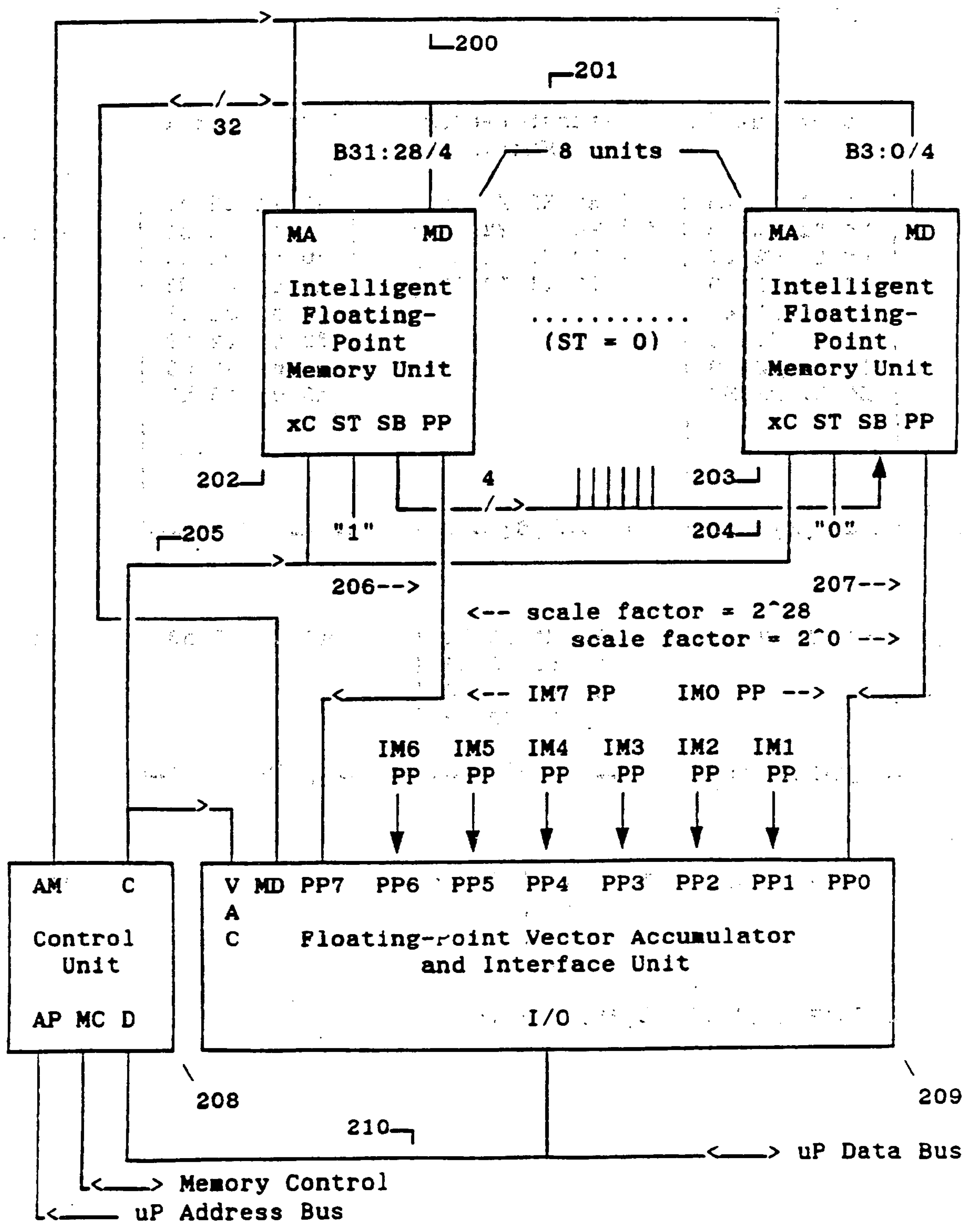
Input Matrix Transformation Matrix Output Matrix

$$\underbrace{\begin{vmatrix} A0 & A1 & A2 & A3 \\ B0 & B1 & B2 & B3 \\ C0 & C1 & C2 & C3 \\ D0 & D1 & D2 & D3 \\ E0 & E1 & E2 & E3 \\ F0 & F1 & F2 & F3 \\ G0 & G1 & G2 & G3 \\ H0 & H1 & H2 & H3 \\ & \vdots & & \end{vmatrix}}_{[A]} * \underbrace{\begin{vmatrix} X0 & X1 & X2 & X3 \\ Y0 & Y1 & Y2 & Y3 \\ Z0 & Z1 & Z2 & Z3 \\ T0 & T1 & T2 & T3 \end{vmatrix}}_{[B]} = \underbrace{\begin{vmatrix} a0 & a1 & a2 & a3 \\ b0 & b1 & b2 & b3 \\ c0 & c1 & c2 & c3 \\ d0 & d1 & d2 & d3 \\ e0 & e1 & e2 & e3 \\ f0 & f1 & f2 & f3 \\ g0 & g1 & g2 & g3 \\ h0 & h1 & h2 & h3 \\ & \vdots & & \end{vmatrix}}_{[C]} \qquad \text{eqn 1.1}$$

$$\underbrace{\begin{vmatrix} A0 & A1 & A2 & A3 \end{vmatrix}}_{[A(Row\ 0)]} * \underbrace{\begin{vmatrix} X0 & X1 & X2 & X3 \\ Y0 & Y1 & Y2 & Y3 \\ Z0 & Z1 & Z2 & Z3 \\ T0 & T1 & T2 & T3 \end{vmatrix}}_{[B]} = \underbrace{\begin{vmatrix} a0 & a1 & a2 & a3 \end{vmatrix}}_{[C(Row\ 0)]} \qquad \text{eqn 1.2}$$

$$\underbrace{\begin{vmatrix} A0 & A1 & A2 & A3 \end{vmatrix}}_{[A(Row\ 0)]} * \underbrace{\begin{vmatrix} X0 \\ Y0 \\ Z0 \\ T0 \end{vmatrix}}_{[B(Col\ 0)]} = a0 \qquad \text{eqn 1.3}$$

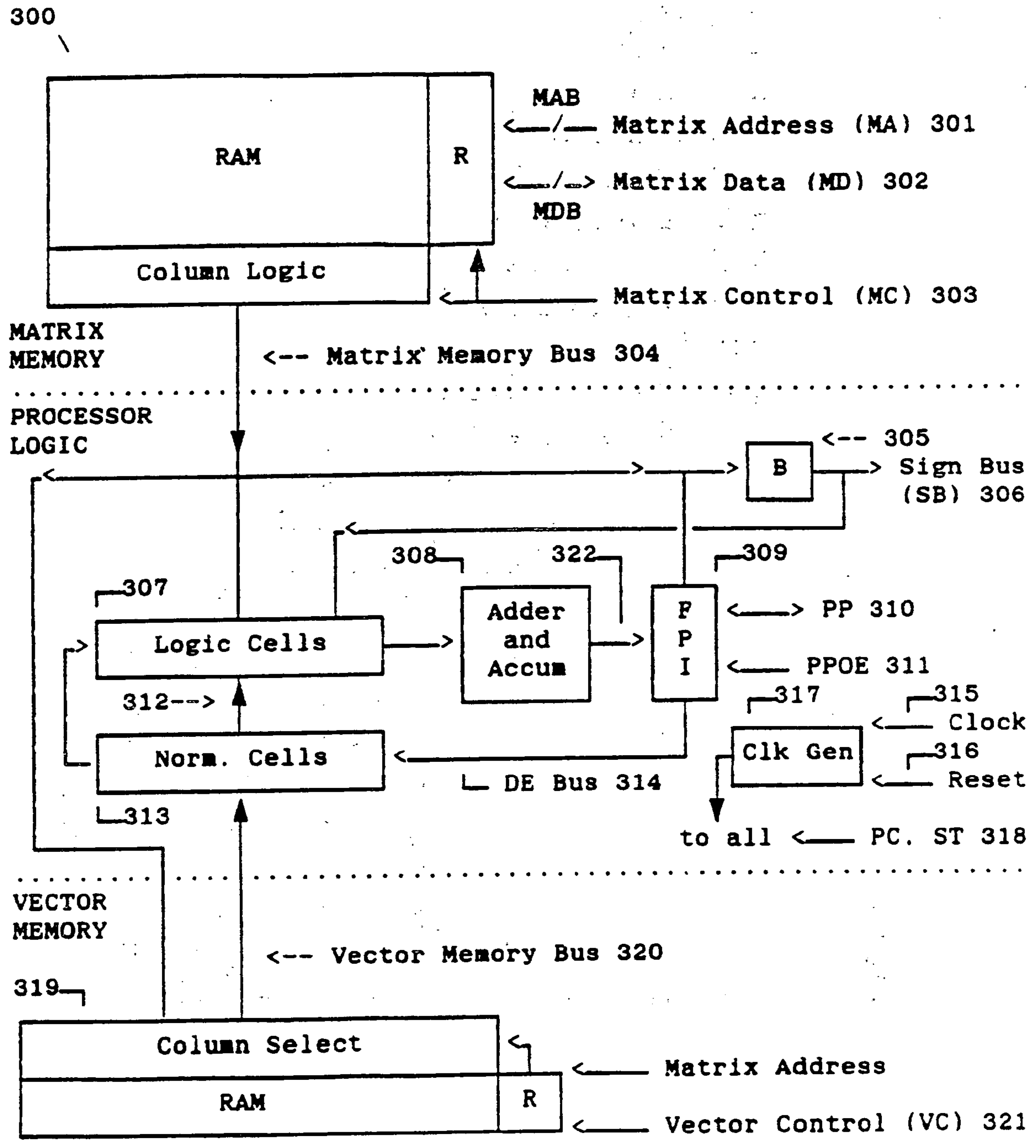
Figure 2



Clock. Reset -> to all

Note: xC = MC. PC and VC

Figure 3


300
RAM
R
Column Logic
MAB
Matrix Address (MA) 301
Matrix Data (MD) 302
MDB
Matrix Control (MC) 303
MATRIX MEMORY
<-- Matrix Memory Bus 304
PROCESSOR LOGIC
<-- 305
B
Sign Bus (SB) 306
308
322
309
307
Logic Cells
Adder and Accum
F P I
PP 310
PPOE 311
312-->
317
315
Clock
Norm. Cells
Clk Gen
316
Reset
DE Bus 314
313
to all
PC. ST 318
VECTOR MEMORY
<-- Vector Memory Bus 320
319
Column Select
RAM
R
Matrix Address
Vector Control (VC) 321

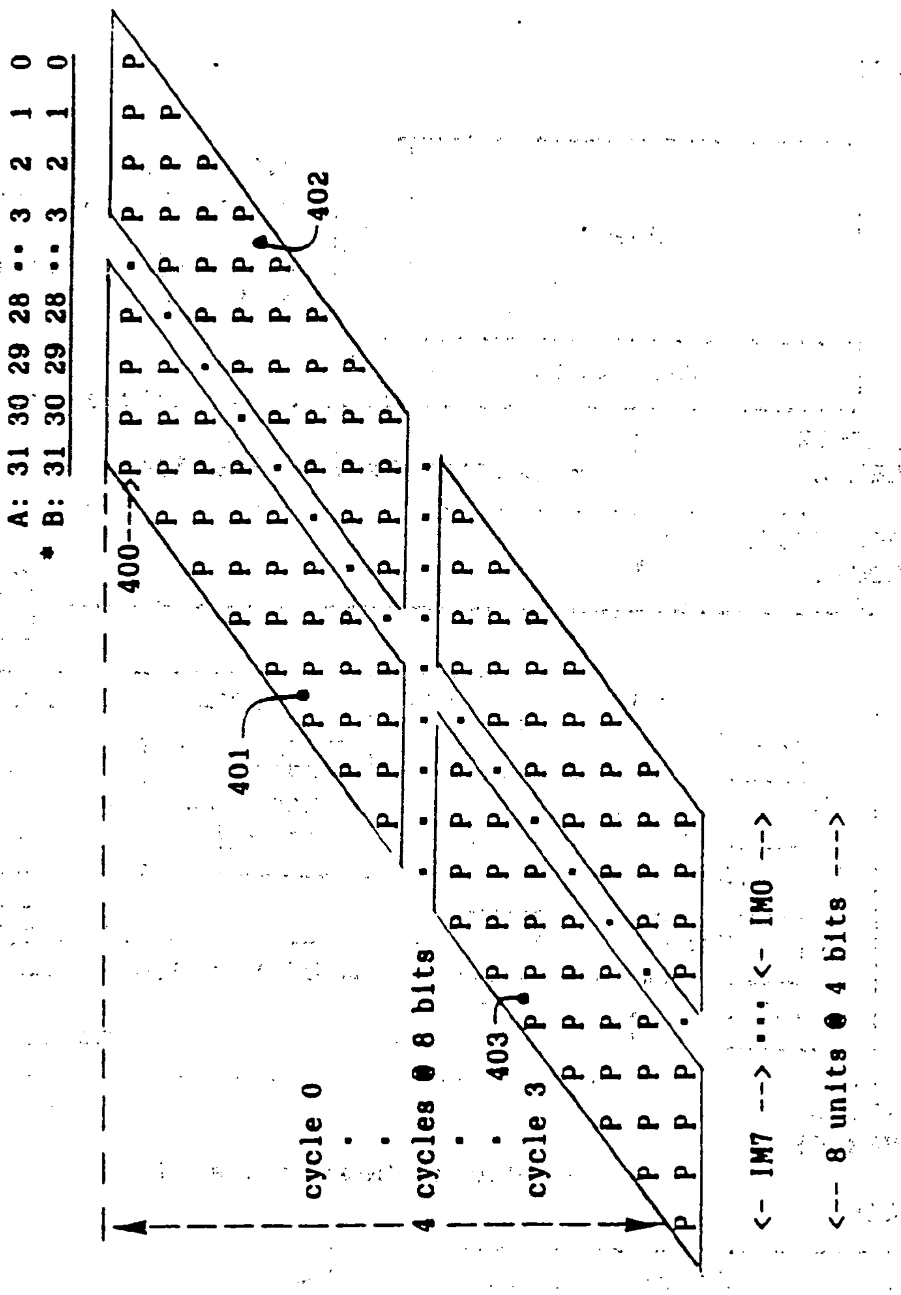

A: 31 30 29 28 .. 3 2 1 0
* B: 31 30 29 28 .. 3 2 1 0
400
401
402
403
cycle 0
4 cycles @ 8 bits
cycle 3
<-- IM7 --> ... <-- IM0 -->
<-- 8 units @ 4 bits -->

Figure 4

Figure 5

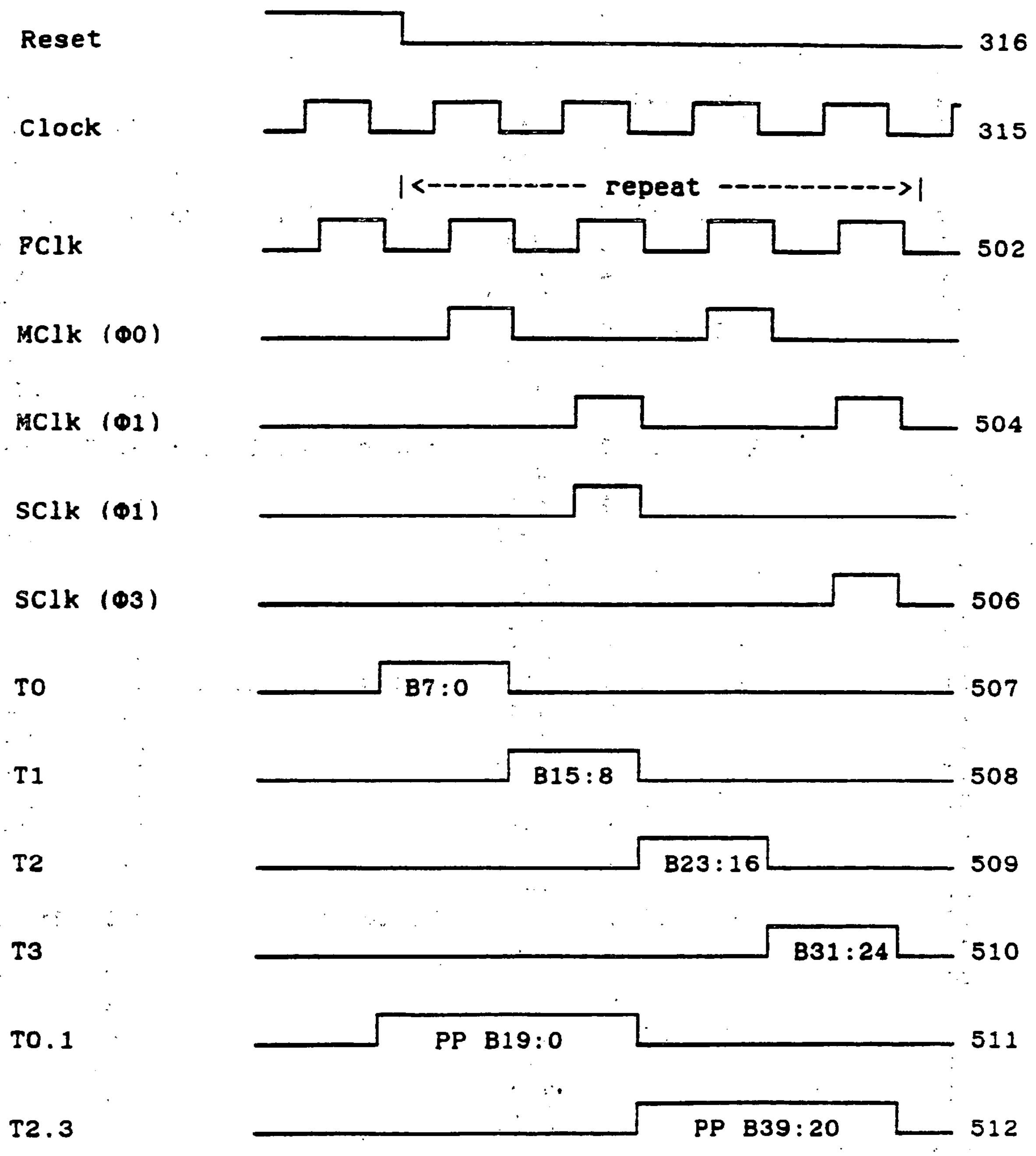

Reset
316
Clock
315
|<---------- repeat ----------->|
FClk
502
MClk (Φ0)
MClk (Φ1)
504
SClk (Φ1)
SClk (Φ3)
506
T0
B7:0
507
T1
B15:8
508
T2
B23:16
509
T3
B31:24
510
T0.1
PP B19:0
511
T2.3
PP B39:20
512

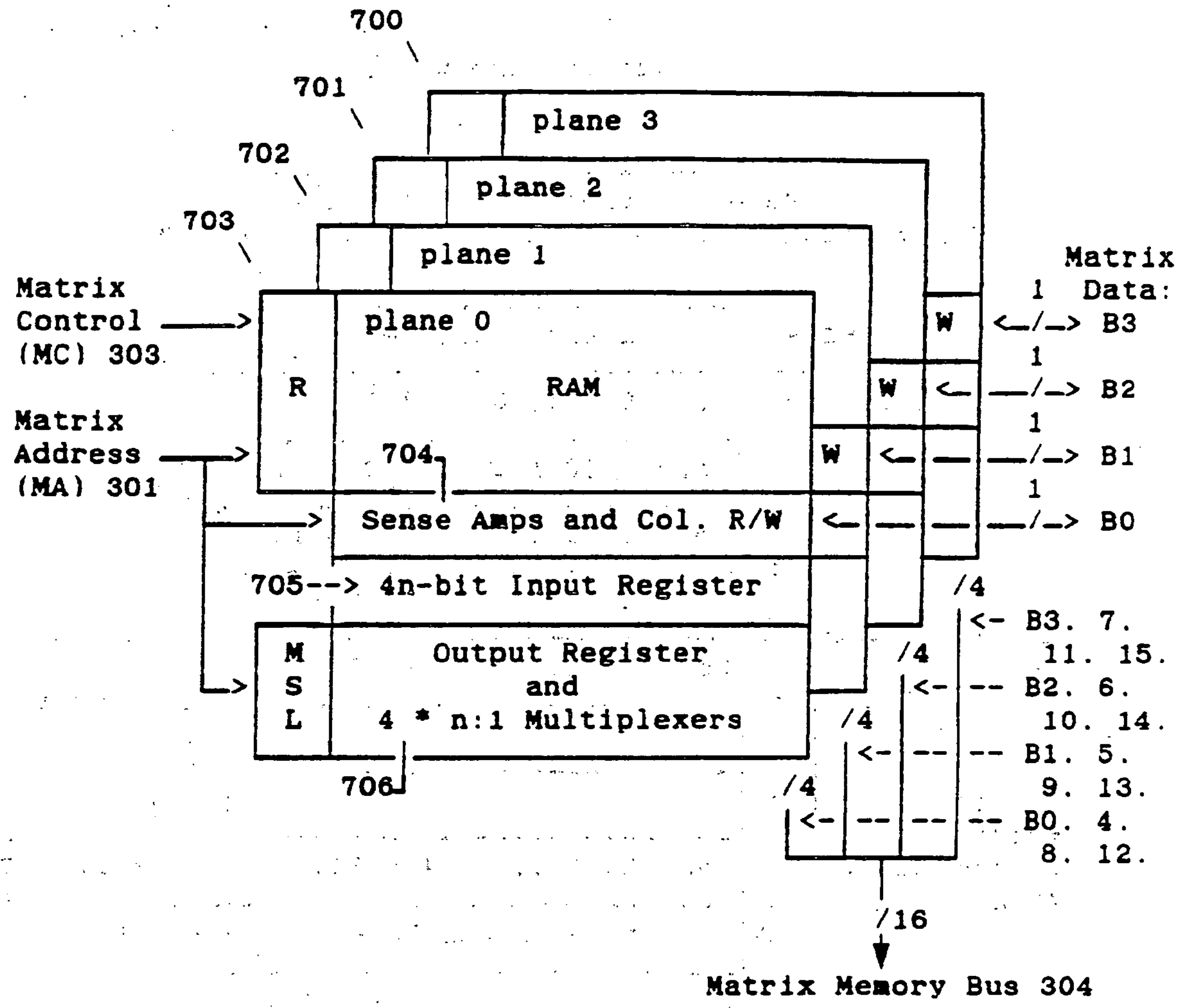
Figure 6

| Memory Column | Matrix Format | | Matrix Memory Bus Bit | Vector Format | | |
|---|---|---|---|---|---|---|
| 0 | Matrix Row K. Col | 0 | M | Vector Col L. Row | 0. Bit | M |
| 1 | | 1 | M+4 | | 1. | M |
| 2 | | 2 | M+8 | | 2. | M |
| 3 | | 3 | M+12 | | 3. | M |
| 4 | K+1. Col | 0 | M | | 0. | M+4 |
| 5 | | 1 | M+4 | | 1. | M+4 |
| 6 | | 2 | M+8 | | 2. | M+4 |
| 7 | | 3 | M+12 | | 3. | M+4 |
| 8 | K+2. Col | 0 | M | Vector Col L. Row | 0. Bit | M+8 |
| 9 | | 1 | M+4 | | 1. | M+8 |
| 10 | | 2 | M+8 | | 2. | M+8 |
| 11 | | 3 | M+12 | | 3. | M+8 |
| 12 | K+3. Col | 0 | M | | 0. | M+12 |
| 13 | | 1 | M+4 | | 1. | M+12 |
| 14 | | 2 | M+8 | | 2. | M+12 |
| 15 | | 3 | M+12 | | 3. | M+12 |
| 16 | K+4. Col | 0 | M | Vector Col L. Row | 0. Bit | M+16 |
| 17 | | 1 | M+4 | | 1. | M+16 |
| 18 | | 2 | M+8 | | 2. | M+16 |
| 19 | | 3 | M+12 | | 3. | M+16 |
| 20 | K+5. Col | 0 | M | | 0. | M+20 |
| 21 | | 1 | M+4 | | 1. | M+20 |
| 22 | | 2 | M+8 | | 2. | M+20 |
| 23 | | 3 | M+12 | | 3. | M+20 |
| 24 | K+6. Col | 0 | M | Vector Col L. Row | 0. Bit | M+24 |
| 25 | | 1 | M+4 | | 1. | M+24 |
| 26 | | 2 | M+8 | | 2. | M+24 |
| 27 | | 3 | M+12 | | 3. | M+24 |
| 28 | K+7. Col | 0 | M | | 0. | M+28 |
| 29 | | 1 | M+4 | | 1. | M+28 |
| 30 | | 2 | M+8 | | 2. | M+28 |
| 31 | | 3 | M+12 | | 3. | M+28 |
| 32 | K+8. Col | 0 | M | Vector Col L+1. Row | 0. Bit | M |
| 33 | | 1 | M+4 | | 1. | M |
| 34 | | 2 | M+8 | | 2. | M |
| 35 | | 3 | M+12 | | 3. | M |
| 36 | K+9. Col | 0 | M | | 0. | M+4 |
| 37 | | 1 | M+4 | | 1. | M+4 |
| 38 | | 2 | M+8 | | 2. | M+4 |
| 39 | | 3 | M+12 | | 3. | M+4 |

etc to 511

Notes: 1. M = plane number = 0 to 3
2. Multiplexer Select = Integer(Memory_Column/4)

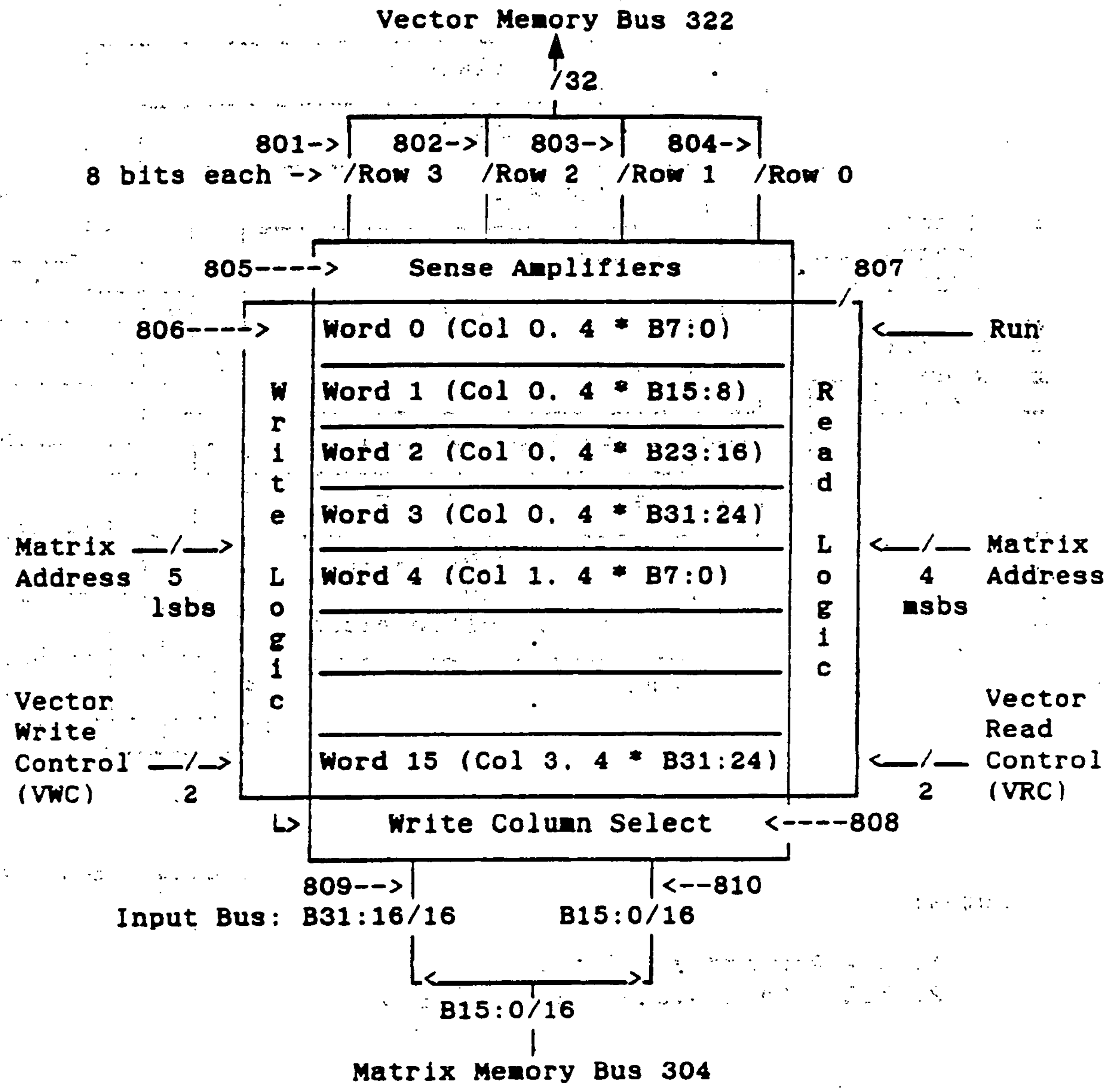
Figure 7



Notes:

1. R = Row address logic
2. MSL = Multiplexer Select Logic

Figure 8



clocks -> to all

Figure 9

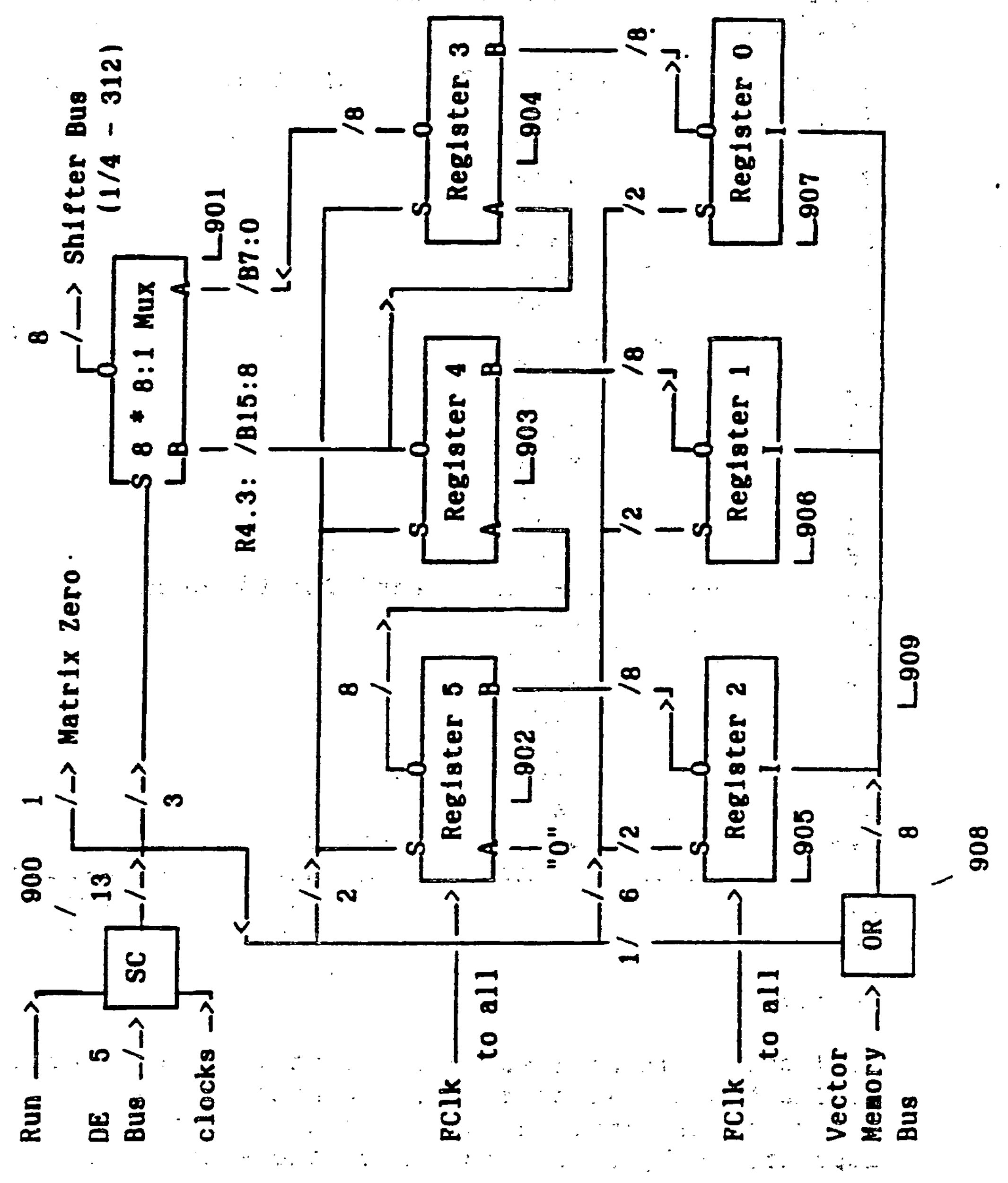

Run
DE Bus
5
clocks
SC
900
13
1
Matrix Zero
3
8
Shifter Bus (1/4 - 312)
8 * 8:1 Mux
S
B
A
O
901
R4.3: /B15:8
/B7:0
Register 5
Register 4
Register 3
Register 2
Register 1
Register 0
902
903
904
905
906
907
2
8
/8
/2
"0"
6
1/
FClk
to all
Vector Memory Bus
OR
908
909

Figure 10

| Delta Exponent | Register Loading | Multiplexer Function (data -> Shifter Bus) |
|---|---|---|
| 0 | Mant{B23:16 -> R2. B15:8 -> R1. B7:0 -> R0}: | R4.3 B7:0 |
| 1 | same | 8:1 |
| 2 | same | 9:2 |
| 3 | same | 10:3 |
| 4 | same | 11:4 |
| 5 | same | 12:5 |
| 6 | same | 13:6 |
| 7 | same | 14:7 |
| 8 | 0 -> R2. Mant{B23:16 -> R1. B15:8 -> R0}: | R4.3 7:0 |
| 9 | same | 8:1 |
| 10 | same | 9:2 |
| 11 | same | 10:3 |
| 12 | same | 11:4 |
| 13 | same | 12:5 |
| 14 | same | 13:6 |
| 15 | same | 14:7 |
| 16 | 0 -> R2. 0 -> R1. Mant{B23:16} -> R0: | R4.3 7:0 |
| 17 | same | 8:1 |
| 18 | same | 9:2 |
| 19 | same | 10:3 |
| 20 | same | 11:4 |
| 21 | same | 12:5 |
| 22 | same | 13:6 |
| 23 | same | 14:7 |
| 24+ | 0 -> R2. 0 -> R1. 0 -> R0: | R1.0 7:0 |
| 29 | additional information: matrix = 0. vector = 0 | |
| 30 | matrix /= 0. vector = 0 | |
| 31 | matrix = 0. vector /= 0 | |

Notes:

1. Above logic applies independently to each Normalizer Cell.
2. The hidden bit of the vector mantissa is set to one by the OR block if the mantissa is not zero. The bit is set when the appropriate byte is passed from the Vector Memory bus to the registers in the Normalizer Cell. No other bits are affected.
3. Mant = Mantissa. "/=" is "not equal".
4. The operation of the registers versus time is:

```
   Cycle 0: R2 -> R5. R1 ->R4. R0 -> R3.
         1: byte of mantissa -> R5. R4 or R3: 0 -> R2 -> R1 -> R0.
         2: byte of mantissa -> R5. R4 or R3: 0 -> R2 -> R1 -> R0.
         3: byte of mantissa -> R5. R4 or R3: 0 -> R2 -> R1 -> R0.
```

5. Delta Exponent is carried by the DE Bus.

Figure 11

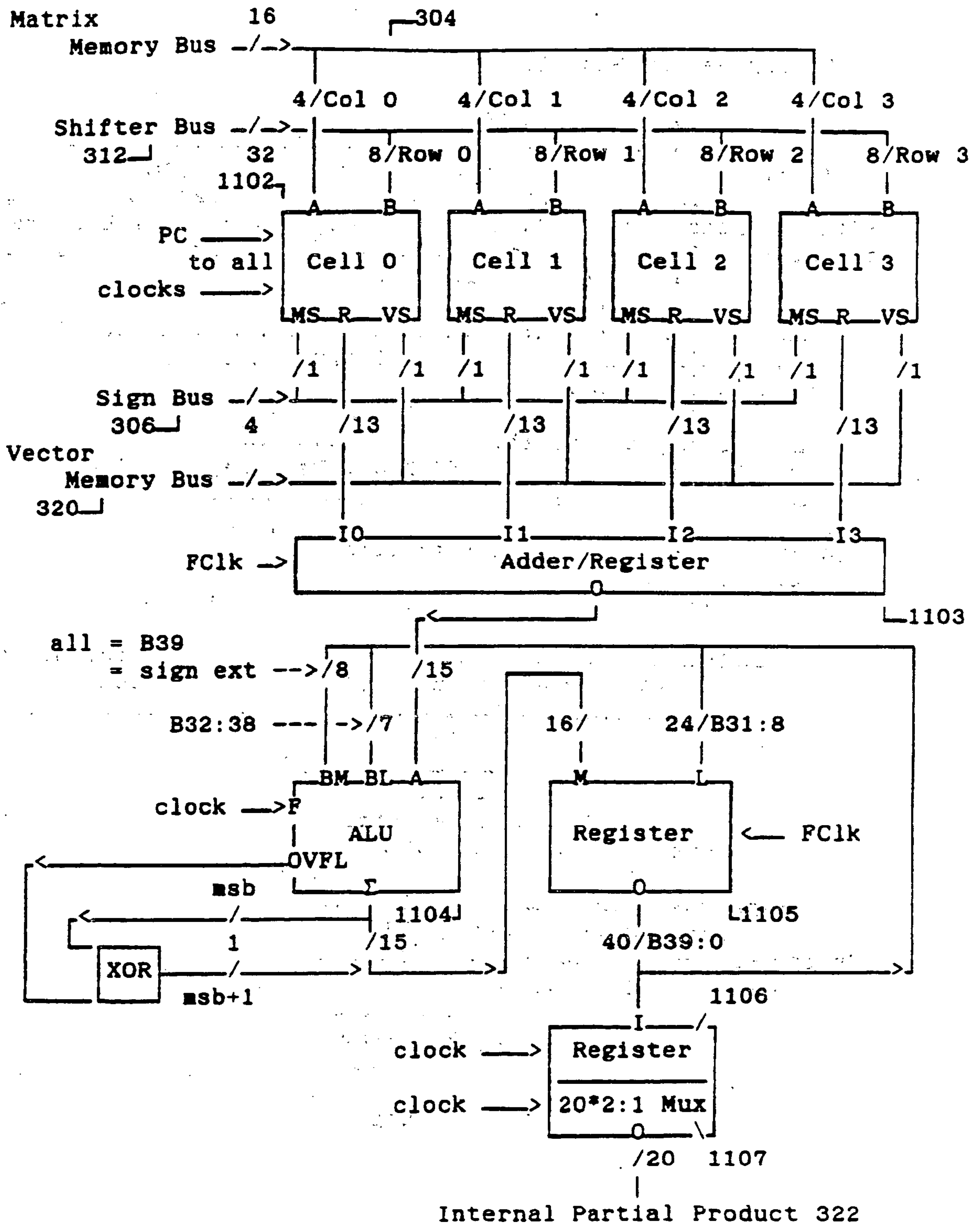

Matrix Memory Bus
16
304
4/Col 0
4/Col 1
4/Col 2
4/Col 3
Shifter Bus
312
32
1102
8/Row 0
8/Row 1
8/Row 2
8/Row 3
A
B
PC
to all
clocks
Cell 0
Cell 1
Cell 2
Cell 3
MS
R
VS
/1
Sign Bus
306
4
/13
Vector Memory Bus
320
I0
I1
I2
I3
FClk
Adder/Register
O
1103
all = B39
= sign ext
/8
/15
B32:38
/7
16/
24/B31:8
BM
BL
A
M
L
clock
F
ALU
OVFL
Register
FClk
msb
Σ
1104
1105
1
/15
40/B39:0
XOR
msb+1
1106
I
clock
Register
20*2:1 Mux
/20
1107
Internal Partial Product 322

Figure 12

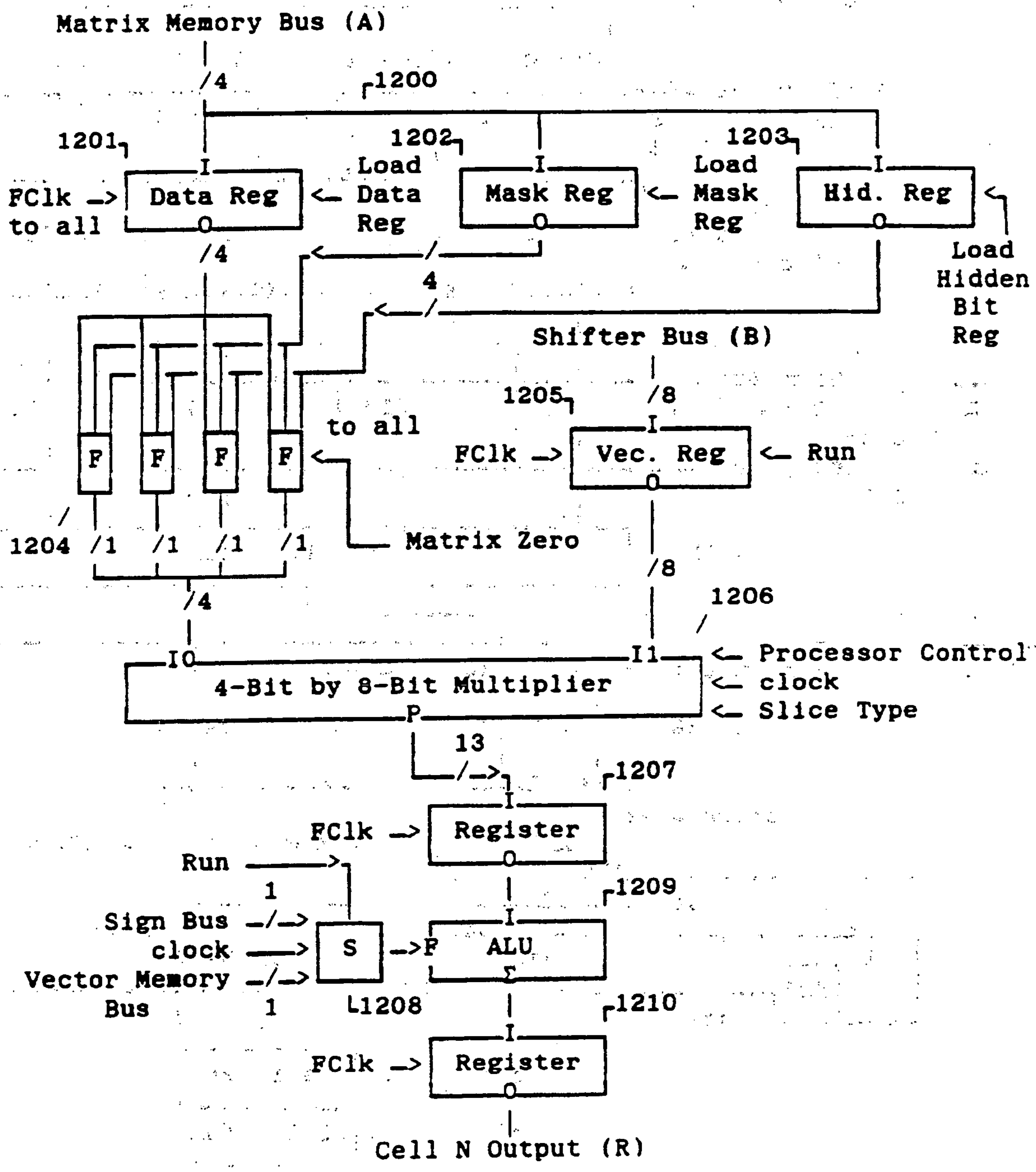

Matrix Memory Bus (A)
/4
1200
1201
1202
1203
I
Data Reg
Mask Reg
Hid. Reg
O
FClk -> to all
Load Data Reg
Load Mask Reg
Load Hidden Bit Reg
/4
4
Shifter Bus (B)
1205
/8
Vec. Reg
FClk ->
<- Run
to all
F
F
F
F
1204
/1
Matrix Zero
/8
1206
/4
I0
I1
4-Bit by 8-Bit Multiplier
P
<- Processor Control
<- clock
<- Slice Type
13
1207
FClk ->
Register
Run
1
1209
Sign Bus
clock
Vector Memory Bus
S
1208
F
ALU
Σ
1210
Register
Cell N Output (R)

Figure 13

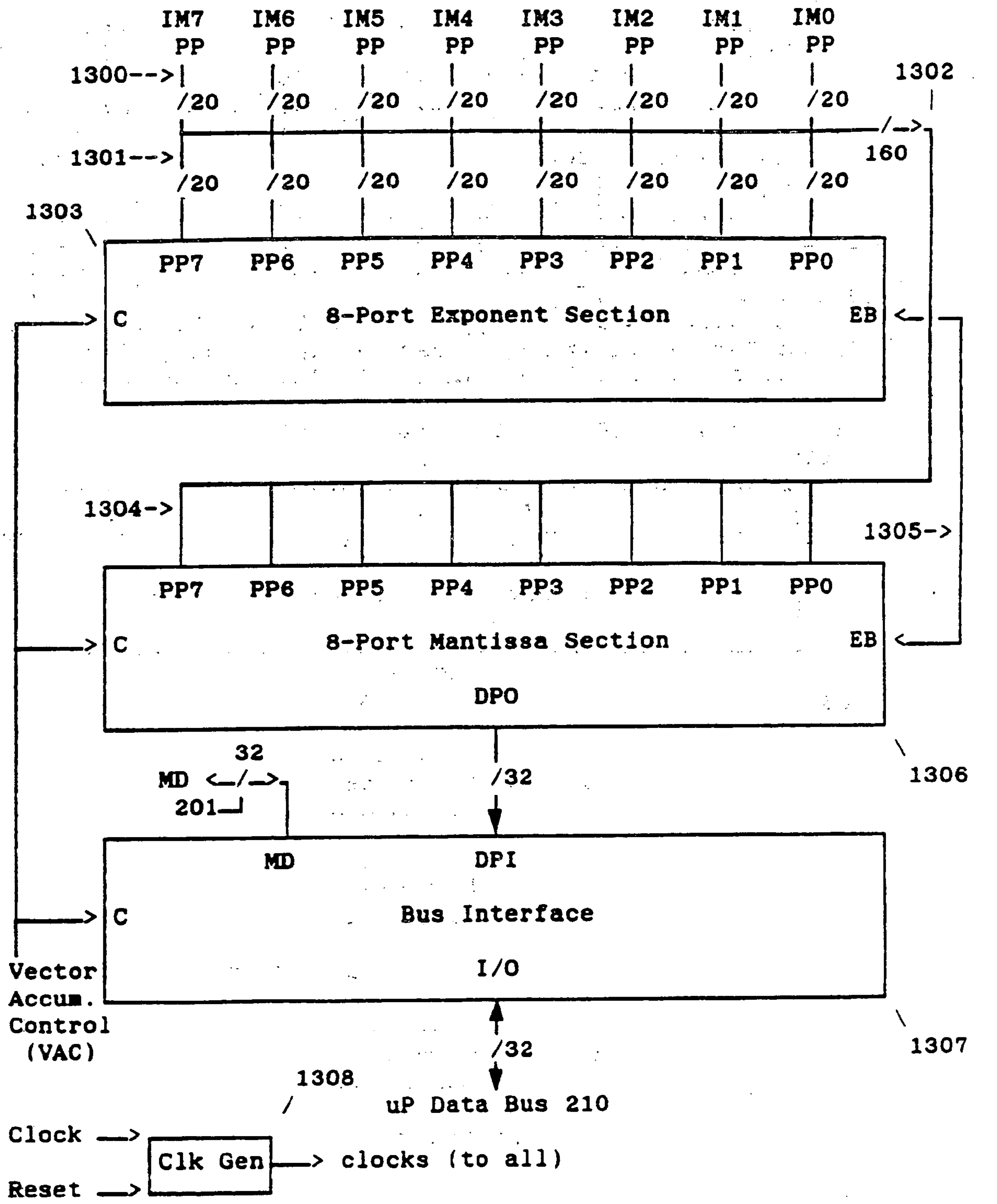

IM7 PP
IM6 PP
IM5 PP
IM4 PP
IM3 PP
IM2 PP
IM1 PP
IM0 PP
1300-->
1301-->
/20
1302
160
1303
PP7
PP6
PP5
PP4
PP3
PP2
PP1
PP0
C
8-Port Exponent Section
EB
1304->
1305->
8-Port Mantissa Section
DPO
1306
32
MD
201
/32
MD
DPI
Bus Interface
I/O
1307
Vector Accum. Control (VAC)
/32
1308
uP Data Bus 210
Clock
Reset
Clk Gen
clocks (to all)

Figure 14

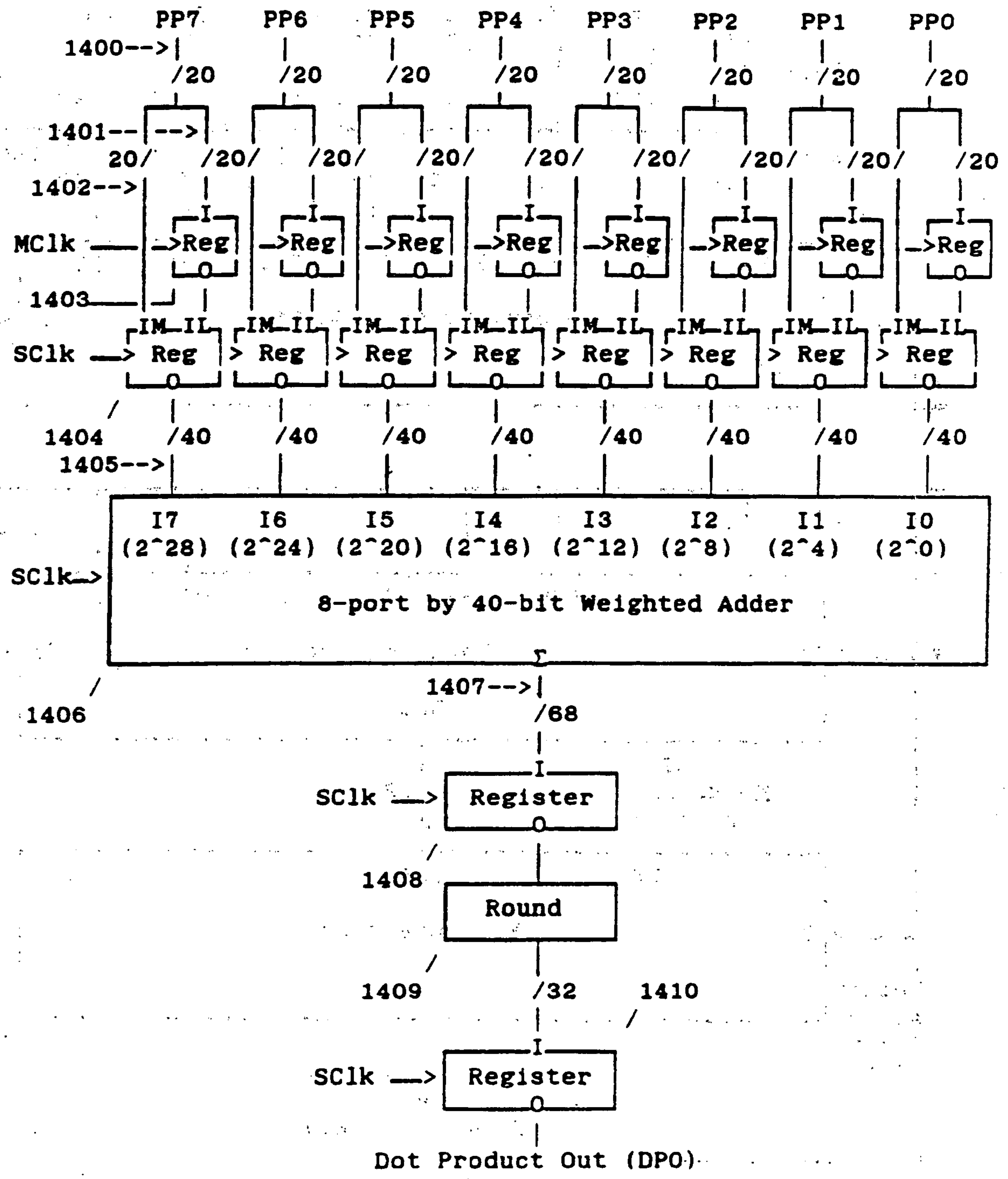
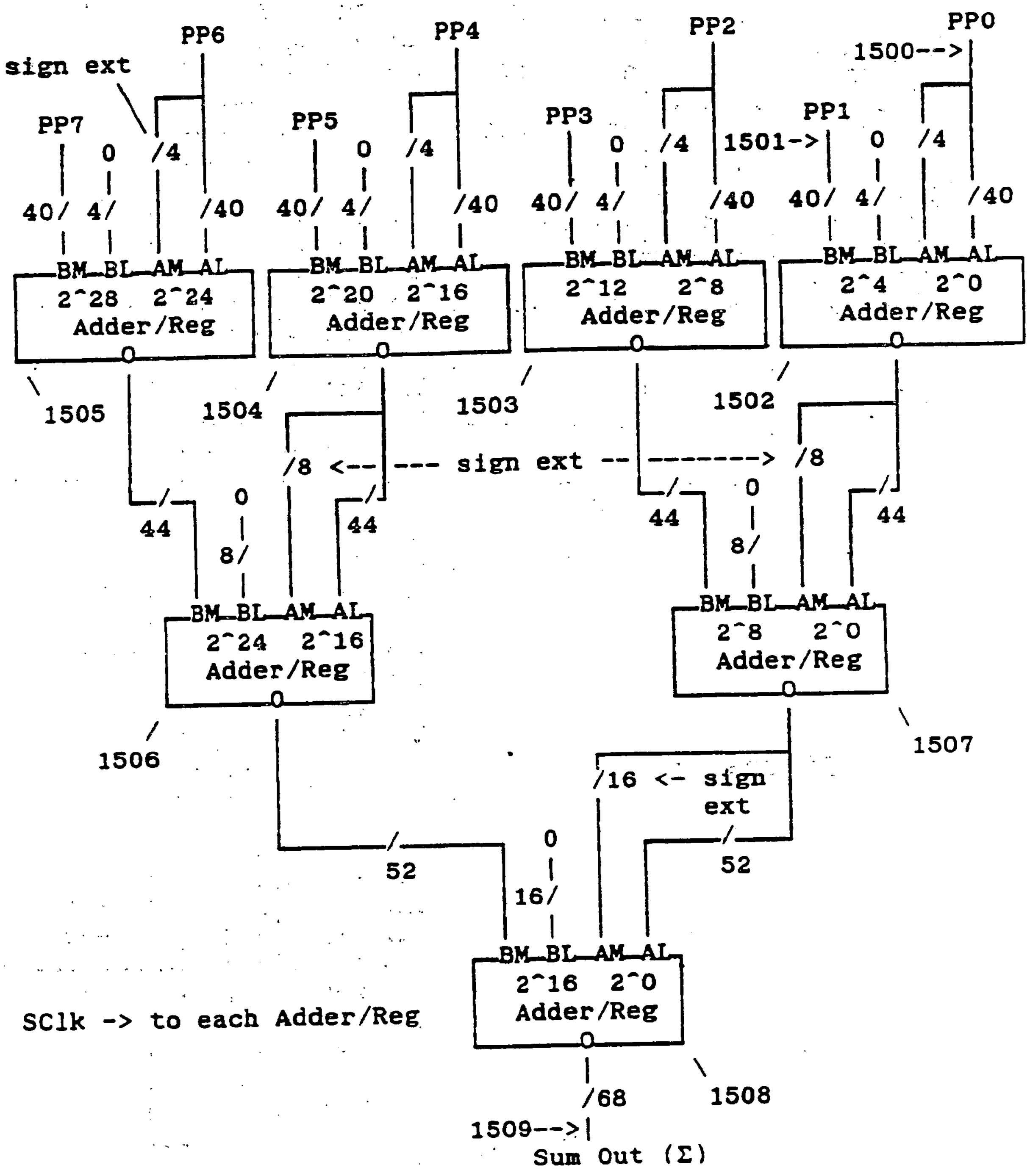
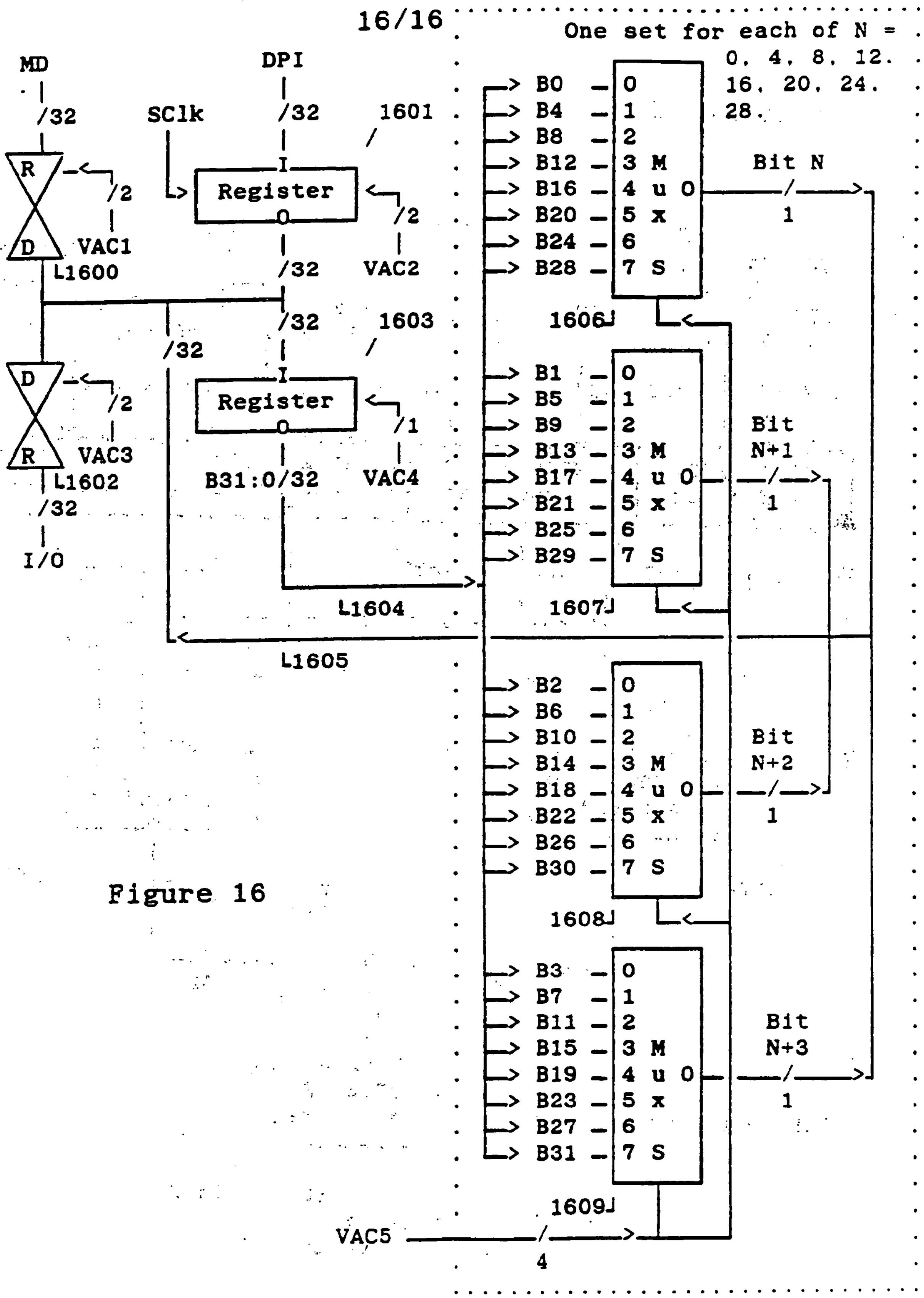

PP7
PP6
PP5
PP4
PP3
PP2
PP1
PP0
1400-->
/20
1401--
-->
20/
/20/
/20
1402-->
I
Reg
O
MClk
1403
IM
IL
SClk
1404
/40
1405-->
I7 (2^28)
I6 (2^24)
I5 (2^20)
I4 (2^16)
I3 (2^12)
I2 (2^8)
I1 (2^4)
I0 (2^0)
8-port by 40-bit Weighted Adder
Σ
1406
1407-->
/68
Register
1408
Round
1409
/32
1410
Dot Product Out (DPO)

Figure 15



SClk -> to each Adder/Reg



One set for each of N = 0, 4, 8, 12. 16, 20, 24. 28.
MD
/32
SClk
DPI
/32
1601
R
D
/2
VAC1
L1600
Register
I
O
/2
VAC2
/32
/32
1603
/32
D
R
/2
VAC3
L1602
/32
I/O
Register
/1
VAC4
B31:0/32
L1604
L1605
B0 B4 B8 B12 B16 B20 B24 B28
B1 B5 B9 B13 B17 B21 B25 B29
B2 B6 B10 B14 B18 B22 B26 B30
B3 B7 B11 B15 B19 B23 B27 B31
0 1 2 3 4 5 6 7
Mux
O
S
1606
1607
1608
1609
Bit N
Bit N+1
Bit N+2
Bit N+3
1
VAC5
4

Figure 16

# INTERNATIONAL SEARCH REPORT

International Application No. PCT/US89/02864

**I. CLASSIFICATION OF SUBJECT MATTER** (if several classification symbols apply, indicate all) 6

According to International Patent Classification (IPC) or to both National Classification and IPC
IPC(4): G06F 15/31, 15/347
U.S. Cl. 364/900

**II. FIELDS SEARCHED**

Minimum Documentation Searched 7

| Classification System | Classification Symbols |
|---|---|
| U.S. | 364/200,900 |

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched 8

**III. DOCUMENTS CONSIDERED TO BE RELEVANT 9**

| Category * | Citation of Document, 11 with indication, where appropriate, of the relevant passages 12 | Relevant to Claim No. 13 |
|---|---|---|
| A | US, A, 4,051,551 (Burroughs Corporation) 27 September 1977 See entire document. | 1 |
| A | US, A, 4,204,208 (Harris Corporation) 20 May 1980 See entire document. | 1 |

* Special categories of cited documents: 10
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

**IV. CERTIFICATION**

| | |
|---|---|
| Date of the Actual Completion of the International Search: 31 August 1989 | Date of Mailing of this International Search Report: 10 OCT 1989 |
| International Searching Authority: ISA/US | Signature of Authorized Officer: Christina M. Eakman |

FURTHER INFORMATION CONTINUED FROM THE SECOND SHEET

V. ☐ OBSERVATIONS WHERE CERTAIN CLAIMS WERE FOUND UNSEARCHABLE[1]

This international search report has not been established in respect of certain claims under Article 17(2) (a) for the following reasons:

1. ☐ Claim numbers , because they relate to subject matter[12] not required to be searched by this Authority, namely:

2. ☐ Claim numbers , because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out[13], specifically:

3. ☐ Claim numbers____________, because they are dependent claims not drafted in accordance with the second and third sentences of PCT Rule 6.4(a).

VI. ☒ OBSERVATIONS WHERE UNITY OF INVENTION IS LACKING[2]

This International Searching Authority found multiple inventions in this international application as follows:

I. Claim 1 drawn to an intelligent floating-point memory; class 364 subclass 900.

II. Claim 2 drawn to a method of computing the dot-product (Continued on attached sheet.)

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims of the international application.

2. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims of the international application for which fees were paid, specifically claims:

3. ☒ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claim numbers: 1.

4. ☐ As all searchable claims could be searched without effort justifying an additional fee, the International Searching Authority did not invite payment of any additional fee.

Remark on Protest

☐ The additional search fees were accompanied by applicant's protest.

☐ No protest accompanied the payment of additional search fees.

Attachment to Form PCT/ISA/210, Part VI.

of vectors in a floating-point format; class 364 subclass 748.

III. Claim 3 drawn to a method for multiplying a row-vector times a column-vector; class 364 subclass 750.5

IV. Claim 4 drawn to a vector accumulator and interface unit; class 364 subclass 736.